The Assistant in C-Advanced

Authors

* Tamer Yousef
* Khalil Rumman

Contents

Acknowledgment

We dedicate this book to our beloved University, The University of Jordan, which was a dream for us,

It came true and now we are students of this university, a day in it is equal to a thousand-page book .we wrote it after we drank from the rivers of the various sciences.

Brothers students, we have done this book to be your reference in the subject of (C advanced programming), and to make it easy to you to understand C advanced programming in all its parts and details.

We take this opportunity to Thank Dr. Hazam Al-Hairy for teaching us C advanced programming and everything he has done for us.

We appreciate your consideration and kind cooperation with us, we deeply appreciate everything you have done for us, we shall be grateful for the consideration you give to this Book.

Students of the University of Jordan:

- Tamer Yousef
- Khalil Rumman

مقدمة

كان المبرمجون قديماً - في الأربعينات من القرن الماضي- يستخدمون لغة الـ Basic لكتابة برامج كبيرة الحجم- صفحات طويلة من الكود- مع إستخدام مبسط للمتغيرات من النوع العام Global و قد كان التعامل مع الكود في ذلك الوقت بالغ الصعوبة حيث أن قراءة الكود و فهمه مسألة معقدة جداً, و بالتالي كان من الصعوبة تعديل أو صيانة الكود - و كما نعلم فإنّ صيانة البرامج والتطوير عليها مسألة مهمة جداً .

في الستينات ظهر مفهوم البرمجة الهيكلية - أو الإجرائية (Programming Structured) حيث تم استخدام أسماء واضحة للمتغيرات مع التفريق بين المتغيرات العامة Global و المتغيرات المحليّة Local و تم تقسيم البرنامج إلى functions ذات مهام محددة و بالتالي أصبح الكود أسهل قراءةً و فهماً و بالتالي أصبحت عملية الصيانة والتطوير على البرامج أسهل . و من أشهر اللغات التي عملت بهذا النهج C و Pascal و Ada و لكن هذه السهولة كانت سهولة نسبية إذا صح التعبير لأنه عندما يتم تعديل جزء من البرنامج فكان ينتج خطأ غير مقصود في مكان آخر من البرنامج لذلك فنحن بحاجة إلى طريقة جديدة للبرمجة تكون فيها عملية فهم الكود و صيانته والتطوير عليه أكثر سهولة.

في بداية الثمانينات قام العالم الدانماركي (بيارني ستروسترب) بتطوير لغة الـ C إلى لغة C++ بحيث أصبحت تُمثل و تحقق مفاهيم البرمجة الكائنية OOP (Object –Oriented Programming) .

البرمجة الكائنية (Object –Oriented Programming) تعتمد استراتيجية جديدة في البرمجة حيث يتم تقسيم البرنامج إلى كائنات objects هذه الكائنات تتفاعل معاً - في إطار البرنامج - لتحقيق الهدف المطلوب ، و كل كائن يحتوي على البيانات الخاصة به مع الـ functions التي تعمل على هذه البيانات في جزء واحد بحيث لا يمكن التعامل مع الكائن إلا من خلال هذه الـ functions .

فبإستخدام الـ OOP نحن نضمن عدم حدوث خطأ غير مقصود في البرنامج عند التعديل في مكان آخر منه. فالبرنامج هو عبارة عن كائنات - أي مثل البرامج المصغرة - كل كائن منفصل في تركيبه عن الكائن الآخر ، لذلك فالبرمجة الكائنية لا تسمح بتطاير البيانات بعيداً عن الـ functions المتعلقة بها و بالتالي أصبح الكود أكثر وضوحاً و اكثر سهولة ، كما أن الصيانة أصبحت أسهل حيث يمكن تعديل جزء من البرنامج - أحد كائناته - بدون التأثير السلبي على الأجزاء الأخرى كما أن بعض المفاهيم المهمة مثل الصفوف (Class) والوراثة (Inheritance) جعلت الكود أصغر بكثير و أكثر متانه مع سهولة إعادة استخدامه وتعتبر لغة C++ من أشهر لغات الـ OOP.

struct

- ما المقصود بالـ struct ؟

struct is a collection of a fixed number of components. The components are called **members.**

- إنَّ كلمة struct محجوزة، فلا يمكن أن نستخدمها كإسم متغير.... .
- إنَّ الفائدة من الـ structs هي جمع أكثر من متغير مختلف في الـ dataType أو متشابه معاً.حيثُ أنَّ هذه المتغيرات نريد أن نستخدمها بكثرة في البرنامج.

فلو افترضنا من أننا نريد عمل برنامج من أجل إدخال و طباعة الاسم و العمر لـ 1000 شخص أي أننا نحتاج إلى تعريف 1000 متغير من نوع string و 1000 متغير من نوع integer وهذا عدد كبير جداً، لذلك نقوم بإستخدام الـ struct حيثُ أننا لن نحتاج إلا لتعريف string واحد و integer واحد.

o struct Syntax (struct definition).

- عند كتابة الـ struct لا يتم حجز موقع لها في الذاكرة (no memory allocated) لأنها (definition) إلا إذا قمنا بتعريف objects من الـ struct عندها سيتم حجز لكل object موقع خاص به في الذاكرة.

لتوضيح النقطة السابقة

في حال قمنا بتعريف متغير اسمه (x) نوعه integer فإنّه سيتم حجز له موقع في الذاكرة بإسم (x) وليس لـ integer. وهنا الـ struct definition يُمثّل نوع المتغير (أي كأنه الـ integer) حيثُ لا يتم حجز موقع في الذاكرة له بل للمتغير من نوع الـ struct و الذي يُسمى بـ object.

- بعد أن تعرفنا على فائدة الـ struct سنتعلم الآن كيف نكتبه **(يكتب قبل الـ () void main وقبل الـ functions)**.

```
ثابت        متغير
struct structName
{
    struct members list
};
```

- حيثُ أنّ الـ struct Member List يقسم إلى ثلاثة أقسام :public و :private و :protected و جميعها كلمات محجوزة، وسنتكلم في هذا الـ chapter عن الـ public و الـ private أما الـ protected فسنكتلم عنه في الـ inheritance.
- تُقسم الـ members الخاصة بالـ struct إلى قسمين،القسم الأول member variables ويكون عبارة عن متغير و من الممكن كتابته في الـ public أو الـ private أو الـ protected
  و القسم الثاني هو member functions ويكون عبارة عن function و من الممكن كتابته في الـ public أو الـ private أو الـ protected.
- الـ public members إما تكون member variables أو member function أو من كِلا النوعين من الممكن إستخدامها في داخل الـ void main أما الـ private members و الـ protected members تكون إما member variable أو member function أو من كِلا النوعين فلا يمكن إستخدامها في داخل الـ void main ويوجد طريقة واحدة لإستخدامها و هي عن طريق الـ public members.

مثال:-

```
struct student
{
public:
    void print();
    void read();
    double avg;
private:
    int stuNo;
    string fName,lName;
};
```

في هذا المثال اسم الـ struct هو student ويحتوي على 6 members منها 4 member variable وهم avg,stuNo,fName,lName ويحتوي على 2 member Function وهما print,read.أما بالنسبة للـ public member functions فيحتوي على 2 وهما print,read, ويحتوي على public member variable واحد و هو avg ويوجد 3 private member variables وهم stuNo,fName,lName.

➢ من الممكن كتابة الـ public و الـ private بأي ترتيب.

مثال:-

1) Classic في هذا المثال قمنا بكتابة الـ public قبل الـ private

```
struct student
{
public:
    void print();
    void read();
    double avg;
private:
    int stuNo;
    string fName,lName;
};
```

2) في هذا المثال قمنا بكتابة الـ private قبل الـ public

```
struct student
{
private:
    int stuNo;
    string fName,lName;
public:
    void print();
    void read();
    double avg;
};
```

3) default members are public

```
struct student
{                    public تعتبر
    void print();
    void read();
    double avg;
private:
    int stuNo;
    string fName,lName;
};
```

- **Function Definition**

➢ لقد تعرّفنا سابقاً على طريقة كتابة الـ struct و لكن لم نتعلم بعد طريقة كتابة الـ definition للـ functions التابع للـ struct.حيثُ يوجد طريقتان لكتابة الـ definition.

➢ الهدف من هذه الـ functions هو الوصول إلى الـ private members و الـ protected members.

▪ الطريقة الأولة :- يُكتَب الـ definition **بعد الـ void main** كالتالي

```
void student::print()
{
    cout<<"ID :"<<stuNo<<" First Name :"<<fName<<" Last Name :"<<lName<<endl;
}
```

هذا عبارة عن definition لـ print حيثُ قمنا بتحديد وظيفته على أن يقوم بطباعة الـ private Members .

```
void student::read()
{
    cin>>stuNo>>fName>>lName;
}
```

هذا عبارة عن definition لـ read حيثُ عن طريقه نسمح للمستخدم بإدخال قيم للـ private Members.

- الطريقة الثانية :- يُكتَب الـ definition في الـ struct كالتالي

```
struct student
{                                   function Definition
    void print(){cout<<"ID :"<<stuNo<<" First Name :"<<fName<<" Last Name :"<<lName<<endl;}
    void read(){cin>>stuNo>>fName>>lName;}
    double avg;
private:
    int stuNo;
    string fName,lName;
};
```

- **Objects Declaration**

➢ لقد انتهينا هكذا من كتابة الـ Struct و لكن لكي نتمكن من إستخدام الـ members الخاص بالـ struct علينا **أولاً** تعريف objects خاصة بالـ struct .**و يتم ذلك في داخل الـ void main ()** كالتالي.

ثابت متغير

structName objectName ;

مثال :-

```
student stu1,stu2;
```

مشابه لـ
student stu1;
student stu2;

➢ في هذا المثال قمنا بعمل 2 (object) من (struct) student و هما stu1 و stu2 .
➢ لكل منهما موقعه الخاص في الذاكرة و member variables خاص بهما أما الـ member function فهي مشتركة لكل الـ objects. أي في حال قمنا بتحديد قيمة لـstuNo الموجود في stu1 فهذا لا يعني بأننا قمنا بتحديد قيمة الـ stuNo لـ stu2.

➢ يوجد طريقة أخرى لتعريف الـ object وهي بكتابته قبل (;) عند كتابة الـ struct .

```
struct student
{
public:
    void print();
    void read (){cin>>stuNo>>fName>>lName;}
    double avg;
private:
    int stuNo;
    string fName,lName;
}stu1,stu2;
```

o **Accessing members (accessing public member)**

- بعد أن قمنا بعمل object . سنتمكن الآن من الوصول إلى الـ members التابعة للـ object

  و يتم ذلك في داخل الـ void main () .

- كما قلنا سابقاً عند تعريف الـ object أصبح لكل object قمنا بعمله member variable خاص به، و لكي نقوم بتغيير member variable لـ object معيّن ، نقوم بالتالي (جميع هذه الـ members عبارة عن public members).

ثابت متغير

```
objectName.memberName ;
```

- يوجد عدة أمور يمكن عملها على *public member variables الـ* كتعيين قيم أو إدخال قيم لها عن طريق المستخدم أو طباعتها.

```
struct student
{
public:
    void print();
    void read();
    int ID;
    string FIRST;
private:
    double salary;
    string LAST;
    void add_2();
}obj1,obj2;
```

- Assigning values for objects members

```
obj1.ID=123;//valid,because it's public
obj1.LAST="Vendetta";//invalid because it's private
obj1.add_2();//invalid,becuase it's private
obj1.print();//valid because it's public
```

- Reading values for objects members

```
cin>>obj1.ID;//valid,because it's public
cin>>obj1.LAST;//invalid because it's private
obj1.read();//valid
```

- Printing values of objects members

```
cout<<obj1.ID;//valid,because it's public
cout<<obj1.LAST;//invalid because it's private
obj1.print();//valid
```

مثال :- في هذا المثال جميع الـ members ستكون public (نستطيع إستخدامها في الـ void main).

```
#include <iostream>
#include <string>
using namespace std;
struct student
{
    string fName,mName;
    int age;
}stu1;
void main()
{
    student stu2;
    stu1.fName="tom";
    stu1.age=12;
    cout<<"Enter the student first, mid Name and his age"<<endl;
    cin>>stu2.fName>>stu2.mName>>stu2.age;
    cout<<stu1.fName<<stu1.mName<<" "<<stu1.age<<"\n"
        <<stu2.fName<<" "<<stu2.mName<<" "<<stu2.age<<endl;
}
```

سنلاحظ عند تشغيل البرنامج بأنّه سيتم طباعة (space) بدلاً من stu1.mName لأننا لم نقم بإدخال قيمة لها.

```
Enter the student first ,mid name and his age
david vendetta 39
Tom 12
david vendetta 39
```

cout
cin

مثال:- في هذا المثال قمنا بعمل function 2 أحدهما لإدخال قيم للـ private و الآخر لطباعة القيم.

```
#include<iostream>
#include<string>
using namespace std;
struct student
{
    void print();
    void read(){cin>>stuNo>>fName>>lName;}
private:
    int stuNo;
    string fName,lName;
}stu;
void main()
{
    cout<<"Enter The student ID,First and Last Name.\n";
    stu.read();
    cout<<"You Entered."<<endl;
    stu.print();
}
void student::print()
{
    cout<<"ID :"<<stuNo<<" First Name :"<<fName
        <<" Last Name :"<<lName<<endl;
}
```

مثال:- في هذا المثال قمنا بعمل function من أجل تخزين قيم في الـ private members وهو set حيثُ نقوم بإرسال 2 parameters الأول integer و الآخر string.و قمنا بعمل function بحيث يقوم بأخذ القيم التي تّم تخزينها في الـ private members ويضعها في متغيرات في داخل الـ main
انظر إلى الـ function definition .

```
#include<iostream>
#include<string>
using namespace std;
struct person
{
    void set(int num,string fN){stuNo=num;fName=fN;}
    void get(int&,string &);
private:
    int stuNo;
    string fName;
}stu;
void main()
{
    cout<<"Enter The person ID and First Name.\n";
    int n;string f;
    cin>>n>>f;
    stu.set(n,f);
    cout<<"The Information After adding 1 to the ID and \'y\' to the name"<<endl;
    int n1;string f1;
    stu.get(n1,f1);
    cout<<"ID :"<<n1<<" First Name :"<<f1<<endl;
}
void person::get(int& nn,string & ff)
{
    nn=stuNo+1;
    ff=fName+"y";
}
```

سيكون الـ output كالتالي

```
Enter The person ID and First Name.
01002
Jim
The Information After adding 1 to the ID and 'y' to the name
ID :1003 First Name :Jimy
Press any key to continue . . .
```

cin

- **Assignment, Arithmetic & Relational operations.(aggregate operations)**

  - Assignment (=)

إنَّ عملية مساواة (object) بآخر من **نفس** الـ struct تعتبر مسموحة

```
#include <iostream>
#include <string>
using namespace std;
struct student
{
    string str;
    int num;
}obj1,obj2,obj3;
int main()
{
    obj1.str="test";
    obj1.num=3;
    obj2=obj1;//agregate assignment operations are allowed
    cout<<obj2.num<<" "<<obj2.str<<endl;

}
```

بعد obj2=obj1 سيتم تخزين كلمة test في str الخاص بـ obj2 كما أنه سيتم تخزين 3 في num الخاص بـ obj2

عند تشغيل البرنامج سيكون الـ output كالتالي.

```
3 test
cout
```

حيثُ أنَّ جملة

```
obj2=obj1;
```

مساوية لـ

```
obj2.str=obj1.str;
obj2.num=obj1.num;
```

للمزيد من المعلومات عن إستخدامات المساواة راجع الـ appendix A صفحة 103

  - Arithmetic (+,-,*,/)

إنَّ العمليات الحسابية على الـ object من نفس الـ struct تعتبر غير مسموحة

مثال:- :- لو افترضنا أنّ obj1 و obj2 من نفس الـ struct.

```
ob3=obj1+obj2;//agregate arithmetic operations are not allowed
```

في هذا المثال نريد أن نجمع الـ member variable الخاصة بـ obj1 مع الـ member variable الخاصة بـ obj2 وهذا غير مسموح به. **سنتخلّص من هذه المشكلة في الـ operator overloading.**

للمزيد من المعلومات عن إستخدامات العمليات الحسابية راجع الـ appendix A صفحة 103

  - Relational (! =,==,<,>,<=,>=)

إنَّ العمليات العلائقية على الـ object من نفس الـ struct تعتبر غير مسموحة

مثال:- لو افترضنا أنّ obj1 و obj2 من نفس الـ struct.

```
if(obj1==obj3)//agregate relational operations are not allowed
```

في هذا المثال نريد أن نتأكد من إنْ كانت الـ member variable الخاصة بـ obj1 مساوية للـ member variable الخاصة بـ obj2 و هذا غير مسموح به. **سنتخلص من هذه المشكلة في الـ operator overloading.**

للمزيد من المعلومات عن إستخدامات العمليات العلائقية راجع الـ appendix A صفحة 103

o Constructor

- لا نستطيع إعطاء قيم إبتدائية للـ member variables.

```
struct student
{
    void set(int ,string );
    void print();
    int stuNo=0;//invalid
private:
    string fName="NULL";//invalid
}stu;
```

- لكي نتخلص من هذه المشكلة سنستخدم بما يُسمى بالـ constructor بحيث يقوم بإعطاء قيم أولية لكل object نقوم بإنشائه

- إنَّ الـ constructor يعمل تلقائياً (without call) عندما نقوم بتعريف object من الـ struct الذي يحتوي على الـ constructor.

- اسم الـ constructor نفس اسم الـ struct الذي يوجد فيه الـ constructor.

- الـ constructor هو function ولكن بدون data Type فهو ليس void function ولا value-returning function .

- من الممكن أن يحتوي الـ struct على أكثر من constructor. ومع ذلك فإنّ لهم نفس الإسم (أي إسم الـ struct) و لكن يجب أن يختلفوا إما بعدد الـ parameters أو بترتيبها.

- عندما يحتوي الـ struct على أكثر من constructor يجب أن تختلف هذه الـ constructor عن بعضها البعض إما بعدد الـ parameters أو بترتيب الـ parameters أما بالنسبة لنوع الـ parameter فهو نفس نوع الـ dataType للـ memberVariables.
  - في حال لم يأخذ الـ constructor أيّ parameter فيُسمّى default constructor.
  - في حال احتوى الـ constructor على parameter فيُسمّى بـ constructor with parameter.

- من الأفضل عندما يحتوي الـ class على constructor with parameters يجب أن يحتوي على default constructor و العكس صحيح.

- إنّ عدد الـ parameters التي يمكن أن يأخذها الـ constructor كحد أقصى هي عدد الـ member variable.(في العادة).

مثال(1):-

في هذا المثال سنلاحظ بأنه يمكن أن نستخدم أكثر من constructor و جميعهم يختلفون عن بعضهم بنوع الـ parameters أو بعددها

لاحظ حتى و لو كان الـ constructor يأخذ parameter واحد فقط فيجب أن نضع في الـ definition قيم للـ member variables التي لا يوجد لها parameter

```
#include<iostream>
#include<string>
using namespace std;
struct student
{
public:
    void print()
{cout<<"ID :"<<stuNo<<" First Name :"<<fName<<" Last Name :"<<lName<<endl;}
    student();//default constructor/line 1
    student(int n) constructor definition
    {stuNo=n;fName="";lName="NULL";}//constracator with 1 parameter/line 2
    student(int n,string s)
    {stuNo=n;fName=s;lName="";}//constructor with 2 paparmeter/line 3
    student(string s,int n)
    {stuNo=n;fName="";lName=s;}//constructor with 2 paparmeter/line 4
    student(int,string,string);//constructor with 3 parameter/line 5
    student(string s){fName=s;stuNo=1;lName="";}//constructor with 1 parameter/line 6
private:
    int stuNo;
    string fName,lName;
};
void main()
{   student stu1;//will use constrctor in line 1
    stu1.print();
    cout<<"########\n";
    student stu2(12,"David");//will use constructor in line 3
    stu2.print();
    cout<<"########\n";
    student stu3(130);//will use constructor in line 2
    stu3.print();
    cout<<"########\n";
    student stu4(2,"vendetta","david");//will use constructor in line 5
    stu4.print();
    cout<<"########\n";
    student stu5("david",11);//will use constructor in line 4
    stu5.print();
    cout<<"########\n";
    student stu6("david");//will use constructor in line 6
    stu6.print();
}
student::student()
{   stuNo=0;
    fName=" ";
    lName=" ";
}
student::student(int num, string str1, string str2)
{
    stuNo=num;
    fName=str2;
    lName=str1;
}
```

سيتم إرسال هذه القيم إلى الـ constructor

constructor definition

سيكون الـ output كالتالي

```
ID :0 First Name :  Last Name :
########
ID :12 First Name :David Last Name :
########
ID :130 First Name : Last Name :NULL
########
ID :2 First Name :david Last Name :vendetta
########
ID :11 First Name : Last Name :david
########
ID :1 First Name :david Last Name :
Press any key to continue . . .
```

مثال (2):-

```
#include<iostream>
#include<string>
using namespace std;
structstudent
{
public:
    void print()
{cout<<"ID :"<<stuNo<<" First Name :"<<fName<<" Last Name :"<<lName<<endl;}
    student()
    (stuNo=5;avg=10.0;fName="N/A";lName="N/A";)//default constructor/line 1
    student(int,double,string,string);//constructor with 3 parameter/line 5
    double avg;
private:
    int stuNo;
    string fName,lName;
);
void main()
{
    student s,s1(67,12.5,"David","Vendetta");
    s.print();
    cout<<"the average is "<<s.avg<<endl<<"####\n";
    s1.print();
    cout<<"the average is "<<s1.avg<<endl;
}
student::student(int num,double m,string str1, string str2)
{
    stuNo=num;
    avg=m;
    fName=str2;
    lName=str1;
}
```

سيكون الـ output كالتالي

```
ID :5 First Name :N/A Last Name :N/A
the average is 10
####
ID :67 First Name :Vendetta Last Name :David
the average is 12.5
Press any key to continue . . .
```

- **Constructor with default parameters**

لقد كتبنا في المثال الأول عدد كبير من الـ constructors لنحاول تغطية جميع الإحتمالات التي يمكن أن نرسلها إلى الـ constructor ومع ذلك لم نقم بتغطية جميع الإحتمالات. لذلك سنستخدم الـ constructor with default parameter لتغطية جميع الإحتمالات التي من الممكن إرسالها للـ constructor.

مثال:-

```
#include<iostream>
#include<string>
using namespace std;
struct student
{
public:
    void print()
{cout<<"ID :"<<stuNo<<",First Name :"<<fName<<",Last Name :"<<lName<<endl;}
    student(int n=0,string s1="",string s2="NUll");//constructor with deafult parameter
private:
    int stuNo;
    string fName,lName;
};
void main()
{
    student s;
    student s1(67,"Vendetta","David");
    student s3(67,"","David");
    s.print();
    cout<<"#########\n";
    s1.print();
    cout<<"#########\n";
    s3.print();
}
//the definition of the constructor
student::student(int n,string s1, string s2)
{
    stuNo=n;
    fName=s2;
    lName=s1;
}
```

لا يحتاج إلى default constructor

سيكون الـ output كالتالي

```
ID :0,First Name :NUll,Last Name :
#########
ID :67,First Name :David,Last Name :Vendetta
#########
ID :67,First Name :David,Last Name :
Press any key to continue . . .
```

يوجد طريقة أخرى لكتابة الكود السابق

يحتاج إلى default constructor

```
#include<iostream>
#include<string>
using namespace std;
structstudent
{
public:
    void print()
{cout<<"ID :"<<stuNo<<" First Name :"<<fName<<" Last Name :"<<lName<<endl;}
    student(){stuNo=0;fName=" ";lName=" ";}
    student(int,string,string);//constructor with deafult parameter
private:
    int stuNo;
    string fName,lName;
};
void main()
{
    student s,s1(67,"David","Vendetta");
    s.print();
    cout<<"#########\n";
    s1.print();

}
//the definition of the constructor
student::student(int num=0,string str1=" ", string str2="NULL")
{
    stuNo=num;
    fName=str2;
    lName=str1;
}
```

- **Input/output.(aggregate operations)**

عملية الـ cin و عملية الـ cout كالتالي غير مسموحة (افترض بأنّ obj1 عبارة object)

```
cin>>obj1;//agregate input/output operations are not allowed
cout<<obj1;
```

و لكن الجمل التالية تعتبر مسموحة

```
cin>>obj1.num>>obj1.str;
cout<<obj1.num<<" "<<obj1.str<<endl;
```

حيثُ أنّ كل من str و num عبارة عن public member variable ولو كانت private member variable فهذه الجمل في داخل الـ main تعتبر غير مسموحة

- مقارنة بين الـ structs والـ arrays

Arrays vs. structs

Aggregate Operation	Array	struct
Arithmetic	No	No
Assignment	No	Yes
Input/output	No (except strings)	No
Comparison	No	No
Parameter passing	By reference only	By value or by reference
Function returning a value	No	Yes

o Arrays in structs

- يمكن كتابة Array في داخل الـ struct .
- يساعدنا هذا في حال كان يحتوي الـ struct على عدد كبير من الـ members لها نفس الـ data type.

مثال:-

```
#include<iostream>
#include<string>
using namespace std;
struct person
{
    string str;
    int num[100];
}per1;
void main()
{
    cout<<"Enter 100 number.\n";
    for(int a=0;a<100;a++)
        cin>>per1.num[a];
}
```

مثال:-

```
#include<iostream>
#include<string>
using namespace std;
struct employee
{
public:
    void print();
    void read();
    employee(int n1=0,int n2=0,string s1="",string s2="",string s3="not exist");
private:
    int num[2];
    string str[3];
};
void main()
{   employee emp;
    emp.read();
    emp.print();

}
employee::employee(int n1,int n2,string s1, string s2,string s3)
{   num[0]=n1;
    num[1]=n1;
    str[0]=s1;
    str[1]=s2;
    str[2]=s3;
}
void employee::read()
{   cout<<"Enter the employee First and Last name\n";
    cin>>str[0]>>str[1];
    cout<<"Enter The employee ID\n";
    cin>>num[0];
    cout<<"Enter The employee salary\n";
    cin>>num[1];
}
void employee::print()
{   cout<<"ID :"<<num[0]
        <<"\nFirst Name :"<<str[0]
        <<"\nLast Name :"<<str[1]
        <<"\nSalary ="<<num[1]
        <<"\nPhone Number :"<<str[2]<<endl;
}
```

سيكون الـ output كالتالي

```
Enter the employee First and Last name
David Vendetta
Enter The employee ID
123
Enter The employee salary
666
ID :123
First Name :David
Last Name :Vendetta
Salary =666
Phone Number :not exist
Press any key to continue . . .
```

emp.read();

emp.print();

o structs in Arrays

- يمكن عمل الـ objects على شكل array
- يساعدنا هذا في حال أردنا عمل عدد كبير من الـ objects.

مثال:-

```
#include<iostream>
#include<string>
using namespace std;
struct example
{
    string str;
    int num;
}
void main()
{
    example v[2];
    cout<<"Enter the name and the age of 2 persons\n";
    for(int a=0;a<2;a++)
        cin>>v[a].str>>v[a].num;
    cout<<"you Entered\n";
    for(int b=0;b<2;b++)
        cout<<"Name :"<<v[b].str<<"\nage ="<<v[b].num<<endl;
}
```

مثال:-

```
#include<iostream>
#include<string>
using namespace std;
struct employee
{
public:
    void print();
    void read();
    employee(int n1=0,int n2=0,string s1="",string s2="",string s3="not exist");
private:
    int num[2];
    string str[3];
};
void main()
{   employee emp[3];
    emp[0].print();
    for(int t=1;t<3;t++)
    {
    emp[t].read();
    emp[t].print();
    }
    system("pause");
}

employee::employee(int n1,int n2,string s1, string s2,string s3)
{   num[0]=n1;
    num[1]=n1;
    str[0]=s1;
    str[1]=s2;
    str[2]=s3;
}
void employee::read()
{   cout<<"Enter the employee First and Last name\n";
    cin>>str[0]>>str[1];
    cout<<"Enter The employee ID\n";
    cin>>num[0];
    cout<<"Enter The employee salary\n";
    cin>>num[1];
}
void employee::print()
{   cout<<"ID :"<<num[0]
        <<"\nFirst Name :"<<str[0]
        <<"\nLast Name :"<<str[1]
        <<"\nSalary ="<<num[1]
        <<"\nPhone Number :"<<str[2]<<endl;
}
```

- **structs within structs (struct can be a member of another struct)**

➢ يمكن أن يحتوي الـ struct على (object) من struct آخر .

2

```
struct NameType
{
    string fName,lName;
};
struct student
{
    NameType name;
    int age,mark;
}stu;
```

1

```
struct student
{
    string fName,lName;
    int age,mark;
}stu;
```

إنَّ رقم 2 أكثر ترتيباً من رقم 1 .حيثُ قمنا بوضع الـ string variables في struct مستقل

هكذا نستطيع التعديل على الـ members الخاص بـ name

```
cin>>stu.age;
stu.name.fName="David";
cout<<stu.name.fName<<" ,age= "<<stu.age<<endl;
```

مثال:-

```
#include<iostream>
#include<string>
using namespace std;
struct NameType
{
    string fName,lName;
};
struct student
{
    NameType name;
    int age,mark;
}stu;
void main()
{
    cout<<"Enter your age .\n";
    cin>>stu.age;
    stu.name.fName="David";
    cout<<"Enter Your Last name .\n";
    cin>>stu.name.lName;
    cout<<stu.name.fName<<" "<<stu.name.lName
        <<" ,age= "<<stu.age<<endl;
}
```

سيكون الـ output كالتالي

```
Enter your age .          cout
39                        cin
Enter Your Last name .
vendetta
David vendetta ,age= 39
```

Consider the following declarations:

```
struct xStruct
{
public:
        void func();
        void print() const;
        xStruct();
        xStruct (int, double);
        int u;
private:
        double w;
};
```

1) How many <u>public member variables</u> does struct xStruct have? `1`

2) Write a C++ statement that declares an object t of type xStruct, and initializes the member variables of t to 20 and 35.5 respectively.

```
xStruct t(20, 35.5);
```

3) Write a C++ statement that prints the values of the member variables of the object t.

```
t.print();
```

4) Write the definition of the member function print that prints the contents of u and w.

```
void xStruct::print() const

{

      cout << u << w;

}
```

5) Write the definition of the constructor with parameters that assigns values to u and w.

```
xStruct:: xStruct (int a, double b)

{

      u = a;

      w = b;

}
```

Q) Write a C++ program using STRUCT to do the following:

Section 1) input first and last name for 5 employees and their salaries.

Section 2) output first and last name for 5 employees and their salaries.

**using Struct within Struct like the following

```
struct nameType
{
string fir;
string las;
};
struct empType
{
nameType name;
double salary;
};
```

Solution:

```
#include<iostream>
#include<string>
using namespace std;
struct name_type
{
    string first;
    string last;
};
struct employee_type
{
    name_type name;
    double salary;
};
void main ()
{
    int a,b;
    employee_type emp_info[5];
    cout<<"********************** Section 1 **********************"<<endl;
    cout<<"Enter The First name then the Last then the Salary for five employees"<<endl;
    for (a=0;a<5;a++)
    {
        cin>>emp_info[a].name.first;
        cin>>emp_info[a].name.last;
        cin>>emp_info[a].salary;
    }
    cout<<"********************** Section 2 **********************"<<endl;
    cout<<"Your Employee's Information"<<endl;
    cout<<"\t \t \t "<<endl;
    cout<<"First Name\tLast Name\tSalary"<<endl;
        for (b=0;b<5;b++)
    {
        cout<<emp_info[b].name.first<<"\t";
        cout<<emp_info[b].name.last<<"\t";
        cout<<emp_info[b].salary<<"\t"<<endl;
    }
}
```

class

- ما المقصود بالـ class ؟

class is collection of a fixed number of components. The components are called **members**.

- إنّ كلمة class محجوزة ، فلا يمكن إستخدامها كإسم متغير ولا كإسم ....
- إنّ الـ class مشابه لـ struct و لكن الإختلاف الوحيد هو أنّ الـ default members في الـ class تعتبر private أما في الـ struct فتعتبر public

o Class syntax (class definition).

- عند كتابة الـ class لا يتم حجز موقع لها في الذاكرة لأنها definition.(no memory allocated)حيثُ يتم حجز موقع فقط للـ objects التابعة للـ class

لتوضيح النقطة السابقة

في حال قمنا بتعريف متغير اسمه (x) نوعه integer فإنّه سيتم حجز له موقع في الذاكرة بإسم (x) وليس لـ integer. وهنا الـ class definition يُمثّل نوع المتغير (أي كأنه الـ integer) حيثُ لا يتم حجز موقع في الذاكرة له بل للمتغير من نوع الـ class و الذي يُسمى بـ object.

- نكتب الـ class **قبل الـ void main و قبل الـ functions**.

```
متغير        ثابت
class className
{
    class members list
};
```

- حيثُ أنّ الـ class Member List يقسم إلى ثلاثة أقسام :public و :private و :protected و جميعها كلمات محجوزة، وسنتكلم في هذا الـ chapter عن الـ public و الـ private أما الـ protected فسنكتلم عنه في الـ chapter التالي.
- تقسم الـ members الخاصة بالـ class إلى قسمين القسم الأول هو member variables ويكون عبارة عن متغير و من الممكن كتابته في الـ public أو الـ private أو الـ protected
  و القسم الثاني هو member function ويكون عبارة عن function و من الممكن كتابته في الـ public أو الـ private أو الـ protected.
- الـ public members إما تكون member variables أو member function أو من كِلا النوعين من الممكن إستخدامها في داخل الـ void main أما الـ private members و الـ protected members تكون إما member variable أو member function أو من كِلا النوعين فلا يمكن إستخدامها في داخل الـ void main ويوجد طريقة واحدة لإستخدامها و هي عن طريق الـ public members.

مثال:-

```
#include<iostream>
using namespace std;
class test
{
public:
    void print();
    void set (int,double);
    void add ();
    int a;
private:
    double b;
    int c;
    void get();
};
void main()
{
```

كما ذكرنا سابقاً بأنه من الممكن إستخدامه في الـ void main هو الـ public members بحيث نستخدمها من أجل الوصول إلى الـ private members

ستذهب كل من القيمتين إلى الـ private members حيثُ سيتم تخزين الـ int في المتغير c و أما الـ double فسيتم تخزينها في المتغير b

في هذا المثال اسم الـ class هو test وله 7 members منها 4 member function وهي get , add , set , print ومنها 3 member variables وهي c , b, a, كما أنّ عدد الـ public member functions هو 3 وهي set , print , add وعدد الـ public member variables هو واحد وهو a وأما عدد الـ private member variables فهو 2 وهي b و c و عدد الـ private member functions هو واحد وهو get .

➢ من الممكن كتابة الـ public و الـ private بأي ترتيب.

مثال:-

1) classic في هذا المثال قمنا بكتابة الـ public قبل الـ private

```
class test
{
public:
    void print();
    void set (int,double);
private:
    double b;
    int c;
};
```

2) في هذا المثال قمنا بكتابة الـ private قبل الـ public

```
class test
{
private:
    double b;
    int c;
public:
    void print();
    void set (int,double);
};
```

3) default members are private

```
class test
{
    double b;
    int c;          تعتبر
public:  private members
    void print();
    void set (int,double);
};
```

- **Function Definition**

➢ لقد تعرّفنا سابقاً على طريقة كتابة الـ class و لكن لم نتعلم بعد طريقة كتابة الـ definition للـ functions التابع للـ class.حيثُ يوجد طريقتان لكتابة الـ definition.

- الطريقة الأولة :- يُكتَب الـ definition **بعد الـ void main** كالتالي

```
void test::print()
{
     cout<<"b ="<<b<<" ,c ="<<c<<endl;
}
```

هذا عبارة عن definition لـ print حيثُ قمنا بتحديد وظيفته على أن يقوم بطباعة الـ private Members .

```
void test::set(int c1, double b1)
{
    c=c1;
    b=b1;
}
```

هذا عبارة عن definition لـ set حيثُ قمنا بتخزين c1 و b1 (قيم من الـ main) في الـ private Members.

- الطريقة الثانية :- يُكتَب الـ definition في الـ class كالتالي

```
class test
{
public:                                  function definition
    void print(){cout<<"b ="<<b<<" ,c ="<<c<<endl;}
    void set (int c1,double b1){c=c1;b=b1;}
private:
    double b;
    int c;
};
```

o Objects Declaration

- لقد قمنا سابقاً بالتعرّف على طريقة كتابة الـ class و كتابة الـ definition للـ function الخاص بالـ class و لكننا لم نقم بعد بتعريف objects .حيثُ يتم ذلك **في داخل الـ void main** كالتالي.
- في الـ C ++ الـ Class objects لها اسمان :
  - Class object
  - Class instance

ثابت متغير

```
className objectName ;
```

مثال:-

```
void main()
{
    test obj1,obj2;

}
```

في هذا المثال قمنا بعمل 2 objects اسمهما obj1 و obj2 ولكل منهما موقعه الخاص في الذاكرة و member variables خاصة بهما حيثُ في حال قمنا بإجراء تعديل على الـ member variables التابع لـ obj1 فهذا لا يعني بأننا قمنا بتعديل الـ member variables الخاص بـ obj2 . أما الـ member function فهي مشتركة لكل الـ objects.
أي في حال قمنا بتحديد قيمة لـ b الموجود في obj1 فهذا لا يعني بأننا قمنا بتحديد قيمة الـ b لـ obj2.

- يوجد طريقة أخرى لتعريف الـ objects ويكون ذلك بكتابتها قبل (;) عند كتابة الـ class.

```
class test
{
public:
    void print();
    void set (int ,double );
private:
    double b;
    int c;
}obj1,obj2;
```

o **Accessing members** (accessing public member)

- بعد أن قمنا بإنشاء objects من الـ class سنتمكن من إستخدام الـ public members ويتم ذلك **في داخل الـ void main** كالتالي
- كما قلنا سابقاً عند تعريف الـ object أصبح لكل object قمنا بعمله member variable خاص به، و لكي نقوم بتغيير member variable لـ object معيّن ، نقوم بالتالي (جميع هذه الـ members عبارة عن public members).

ثابت متغير

```
objectName.memberName ;
```

- يوجد عدة أمور يمكن عملها على *public member variables الـ* كتعيين قيم أو إدخال قيم لها عن طريق المستخدم أو طباعتها.

```
class employee
{
public:
    void print();
    void read();
    int ID;
    string FIRST;
private:
    double salary;
    string LAST;
    void add_2();
}obj1,obj2;
```

- Assigning values for objects members

```
obj1.ID=123;//valid,because it's public
obj1.LAST="Vendetta";//invalid because it's private
obj1.add_2();//invalid,becuase it's private
obj1.print();//valid because it's public
```

- Reading values for objects members

```
cin>>obj1.ID;//valid,because it's public
cin>>obj1.LAST;//invalid because it's private
obj1.read();//valid
```

- Printing values of objects members

```
cout<<obj1.ID;//valid,because it's public
cout<<obj1.LAST;//invalid because it's private
obj1.print();//valid
```

مثال:-

```
#include<iostream>
using namespace std;
class test
{
public:
    void print();
    void set (int ,double );
private:
    double b;
    int a;
};
void main()
{
    test obj1,obj2;
    int aa;
    double bb;
    cout<<"Enter 2 Numbers.\n";
    cin>>aa>>bb;
    obj1.set(aa,bb);
    obj1.print();
    aa+=5;
    bb+=5;
    obj2.set(aa,bb);
    obj2.print();
 }
void test::print()
{
    cout<<"a ="<<a<<" ,b ="<<b<<endl;
}
void test::set(int a1,double b1)
{
    a=a1;
    b=b1;
}
```

سيكون الـ output كالتالي

```
Enter 2 Numbers.
6                cin
6.5              cout
a =6 ,b =6.5
a =11 ,b =11.5
```

مثال:-

```
#include<iostream>
using namespace std;
class test
{
    double b;
    int a;
public:
    void print();
    void set (int a1 ,double b1 ){a=a1;b=b1;}
    void add5(){a+=5;b+=6;}
}obj1,obj2;
void main()
{
    int aa;
    double bb;
    cout<<"Enter 2 Numbers.\n";
    cin>>aa>>bb;
    obj1.set(aa,bb);
    obj1.print();
    obj2.set(aa,bb);
    obj2.add5();
    obj2.print();
}
void test::print()
{
    cout<<"a ="<<a<<" ,b ="<<b<<endl;
}
```

لو قمنا بإدخال نفس القيم السابقة فسيكون لدينا نفس الـ output .

مثال:-

```
#include<iostream>
using namespace std;
class test
{
    double b;
    int a;
public:
    void print();
    void print_char(){cout<<"You Entered :"<<s<<endl;}
    void set (int a1 ,double b1 ){a=a1;b=b1;}
    void set_char(char s1){s=s1;}
    char s;
}obj1;
void main()
{
    char d;
    cout<<"Enter a letter.\n";
    cin>>d;
    obj1.set_char(d);
    obj1.print_char();

}
void test::print()
{
    cout<<"a ="<<a<<" ,b ="<<b<<endl;
}
```

سيكون الـ output كالتالي

```
Enter a letter.
w
You Entered :w
```

مثال:-

```
#include<iostream>
#include<string>
using namespace std;
class employee
{
public:
    void print()
    {cout<<"Name :"<<name<<endl<<"Salaray :"<<salary<<endl;}
    void read ();
private:
    string name;
    double salary;
};
void main()
{
    employee p1;
    p1.read();
    p1.print();
}
void employee::read()
{
    cout<<"plz enter the employee name"<<endl;
    cin>>name;
    cout<<"plz enter the employee salary"<<endl;
    cin>>salary;
}
```

سيكون الـ output كالتالي

```
plz enter the employee name
Taner
plz enter the employee salary
500
Name : Taner
Salary : 500
Press any key to continue . . .
```

- **Assignment, Arithmetic & Relational operations.(aggregate operations)**

  - Assignment (=)

العملية الوحيدة المسموحة على الـ objects من نفس الـ class هي الـ (=) .

مثال:- لو افترضنا بأنّ obj1 و obj2 من نفس الـ class.

```
obj1=obj2;//valid
```

سيتم تخزين القيم الموجودة في الـ member variables الخاصة بـ obj2 في obj1

مثال:-

```
#include<iostream>
#include<string>
using namespace std;
class student
{
public:
    void print()
{cout<<"ID :"<<ID<<"\n Name :"<<name<<"\nSalary ="<<ID<<endl;}
    void read();
    int ID;
private:
    double salary;
    string name;
}obj1,obj2;
void main()
{
    cout<<"Enter The employee ID\n";
    cin>>obj2.ID;
    obj2.read();
    obj1=obj2;//valid
    obj1.print();
}

void student::read()
{
    cout<<"Enter the employee name\n";
    cin>>name;
    cout<<"Enter The employee salary\n";
    cin>>salary;
}
```

بعد أن قمنا بتخزين قيم للـ member variables الخاصة بـ obj2 قمنا بوضعها في obj1 و طباعتها

للمزيد من المعلومات عن إستخدامات المساواة راجع الـ appendix B صفحة 105

- Arithmetic (+,-,*,/)

إنَّ العمليات الحسابية على الـ object من نفس الـ class تعتبر غير مسموحة

مثال:- :- لو افترضنا أنّ obj1 و obj2 من نفس الـ class.

```
ob3=obj1+obj2;//agregate arithmetic operations are not allowed
```

في هذا المثال نريد أن نجمع الـ member variable الخاصة بـ obj1 مع الـ member variable الخاصة بـ obj2 وهذا غير مسموح به. سنتخلّص من هذه المشكلة في الـ operator overloading.

للمزيد من المعلومات عن إستخدامات العمليات الحسابية راجع الـ appendix B صفحة 105

- Relational (! =,==,<,>,<=,>=)

إنَّ العمليات العلائقية على الـ object من نفس الـ class تعتبر غير مسموحة

مثال:- لو افترضنا أنّ obj1 و obj2 من نفس الـ class.

```
if(obj1==obj3)//agregate relational operations are not allowed
```

في هذا المثال نريد أن نتأكد من إنْ كانت الـ member variable الخاصة بـ obj1 مساوية للـ member variable الخاصة بـ obj2 و هذا غير مسموح به. سنتخلص من هذه المشكلة في الـ operator overloading.

للمزيد من المعلومات عن إستخدامات العمليات العلائقية راجع الـ appendix B صفحة 105

مثال:-

```
class employee
{
public:
    void read();
    void print();
private:
    string name;
    double salary;
}
void main()
{
    employee x;
    employee y;

    if (x==y)
        cout<< "equal"<<endl;
    else
        cout<<"not equal"<<endl;

    x=y+1;
    y=x*x;
}
```

We can't use realtional operators on class object unless the operators overloaded

we can't use Arithmetic operators on class object

o Constructor

- لا نستطيع إعطاء قيم إبتدائية للـ member variables التي في داخل الـ class.

```
class person
{
public:
    void set(int ,string );
    void print();
    int stuNo=0;//invalid
private:
    string fName="NULL";//invalid
}stu;
```

- لكي نتخلص من هذه المشكلة سنستخدم بما يُسمى بالـ constructor بحيث يقوم بإعطاء قيم أولية لكل object نقوم بإنشائه

- إنَّ الـ constructor يعمل تلقائياً (without call) عندما نقوم بتعريف object من الـ class الذي يحتوي على الـ constructor.

- اسم الـ constructor نفس اسم الـ class الذي يوجد فيه الـ constructor.

- الـ constructor هو function ولكن بدون data Type فهو ليس void function ولا value-returning function .

- من الممكن أن يحتوي الـ class على أكثر من constructor. ومع ذلك فإنّ لهم نفس الإسم (أي إسم الـ class) و لكن يجب أن يختلفوا إما بعدد الـ parameters أو بترتيبها.

- عندما يحتوي الـ class على أكثر من constructor يجب أن تختلف هذه الـ constructor عن بعضها البعض إما بعدد الـ parameters أو بترتيب الـ parameters أما بالنسبة لنوع الـ parameter فهو نفس نوع الـ dataType للـ memberVariables.
  - في حال لم يأخذ الـ constructor أيّ parameter فيُسمّى default constructor.
  - في حال احتوى الـ constructor على parameter فيُسمّى بـ constructor with parameter.

- من الأفضل عندما يحتوي الـ class على constructor with parameters يجب أن يحتوي على default constructor والعكس صحيح.

- إنّ عدد الـ parameters التي يمكن أن يأخذها الـ constructor كحد أقصى هي عدد الـ member variable. (في العادة).

مثال(1) :- في هذا المثال سنلاحظ بأنه يمكن أن نستخدم أكثر من constructor و جميعهم يختلفون عن بعضهم بنوع الـ parameters أو بعددها

لاحظ حتى و لو كان الـ constructor يأخذ parameter واحد فقط فيجب أن نضع في الـ definition قيم للـ member variables التي لا يوجد لها parameter

```
#include<iostream>
#include<string>
using namespace std;
 class student
{
public:
    void print()
{cout<<"ID :"<<stuNo<<" First Name :"<<fName<<" Last Name :"<<lName<<endl;}
    student();//default constructor/line 1
    student(int n) constructor definition
    {stuNo=n;fName="";lName="NULL";}//constracator with 1 parameter/line 2
    student(int n,string s)
    {stuNo=n;fName=s;lName="";}//constructor with 2 paparmeter/line 3
    student(string s,int n)
    {stuNo=n;fName="";lName=s;}//constructor with 2 paparmeter/line 4
    student(int,string,string);//constructor with 3 parameter/line 5
    student(string s){fName=s;stuNo=1;lName="";}//constructor with 1 parameter/line 6
private:
    int stuNo;
    string fName,lName;
};

void main()
{   student stu1;//will use constrctor in line 1
    stu1.print();
    cout<<"########\n";
    student stu2(12,"David");//will use constructor in line 3
    stu2.print();
    cout<<"########\n";
    student stu3(130);//will use constructor in line 2
    stu3.print();
    cout<<"########\n";
    student stu4(2,"vendetta","david");//will use constructor in line 5
    stu4.print();
    cout<<"########\n";
    student stu5("david",11);//will use constructor in line 4
    stu5.print();
    cout<<"########\n";
    student stu6("david");//will use constructor in line 6
    stu6.print();
}
student::student()
{   stuNo=0;
    fName=" ";
    lName=" ";
}
student::student(int num, string str1, string str2)
{
    stuNo=num;
    fName=str2;
    lName=str1;
}
```

سيتم إرسال هذه القيم إلى الـ constructor

constructor definition

سيكون الـ output كالتالي

```
ID :0 First Name :  Last Name :
########
ID :12 First Name :David Last Name :
########
ID :130 First Name : Last Name :NULL
########
ID :2 First Name :david Last Name :vendetta
########
ID :11 First Name : Last Name :david
########
ID :1 First Name :david Last Name :
Press any key to continue . . .
```

مثال (2) :-

```
#include<iostream>
#include<string>
using namespace std;
class student
{
public:
    void print()
{cout<<"ID :"<<stuNo<<" First Name :"<<fName<<" Last Name :"<<lName<<endl;}
    student()
    {stuNo=5;avg=10.0;fName="N/A";lName="N/A";}//default constructor/line 1
    student(int,double,string,string);//constructor with 3 parameter/line 5
    double avg;
private:
    int stuNo;
    string fName,lName;
};
void main()
{
    student s,s1(67,12.5,"David","Vendetta");
    s.print();
    cout<<"the average is "<<s.avg<<endl<<"####\n";
    s1.print();
    cout<<"the average is "<<s1.avg<<endl;
}
student::student(int num,double m,string str1, string str2)
{
    stuNo=num;
    avg=m;
    fName=str2;
    lName=str1;
}
```

سيكون الـ output كالتالي

```
ID :5 First Name :N/A Last Name :N/A
the average is 10
####
ID :67 First Name :Vendetta Last Name :David
the average is 12.5
Press any key to continue . . .
```

- **Constructor with default parameters**

لقد كتبنا في المثال الأول عدد كبير من الـ constructors لنحاول تغطية جميع الإحتمالات التي يمكن أن نرسلها إلى الـ constructor ومع ذلك لم نقم بتغطية جميع الإحتمالات. لذلك سنستخدم الـ constructor with default parameter لتغطية جميع الإحتمالات التي من الممكن إرسالها للـ constructor.

مثال:-

يحتاج إلى default constructor

```
#include<iostream>
#include<string>
using namespace std;
class student
{
public:
    void print()
{cout<<"ID :"<<stuNo<<" First Name :"<<fName<<" Last Name :"<<lName<<endl;}
    student(){stuNo=0;fName=" ";lName=" ";}
    student(int,string,string);//constructor with deafult parameter
private:
    int stuNo;
    string fName,lName;
};
void main()
{
    student s,s1(67,"David","Vendetta");
    s.print();
    cout<<"#########\n";
    s1.print();

}
//the definition of the constructor
student::student(int num=0,string str1=" ", string str2="NULL")
{
    stuNo=num;
    fName=str2;
    lName=str1;
}
```

سيكون الـ output كالتالي

```
ID :0,First Name :NUll,Last Name :
#########
ID :67,First Name :David,Last Name :Vendetta
#########
ID :67,First Name :David,Last Name :
Press any key to continue . . .
```

يوجد طريقة أخرى لكتابة الكود السابق

لا يحتاج إلى default constructor

```
#include<iostream>
#include<string>
using namespace std;
class  student
{
public:
    void print()
{cout<<"ID :"<<stuNo<<",First Name :"<<fName<<",Last Name :"<<lName<<endl;}
    student(int n=0,string s1="",string s2="NUll");//constructor with deafult parameter
private:
    int stuNo;
    string fName,lName;
};
void main()
{
    student s;
    student s1(67,"Vendetta","David");
    student s3(67,"","David");
    s.print();
    cout<<"#########\n";
    s1.print();
    cout<<"#########\n";
    s3.print();
}
//the definition of the constructor
student::student(int n,string s1, string s2)
{
    stuNo=n;
    fName=s2;
    lName=s1;
}
```

o Arrays in classes

- من الممكن أن يحتوي الـ class على array كـ memberVariable

مثال:-

```
#include<iostream>
using namespace std;
class example
{
public:
    void set(int t1[]);
    void print(){for(int c=0;c<5;c++)cout<<p[c]+1<<" ";}
private:
    int p[5];
}obj1;
void main()
{
    int t[5];
    cout<<"Enter 5 Numbers\n";
    for (int a=0;a<5;a++)
        cin>>t[a];
    obj1.set(t);
    cout<<"After adding 1 to that numbers\n";
    obj1.print();
}
void example::set(int tt[])
{
    for(int i=0;i<5;i++)
        p[i]=tt[i];
}
```

سيكون الـ output كالتالي

```
Enter 5 Numbers
76                                  cin
67
88
-88
0
After adding 1 to that numbers
77 68 89 -87 1
```

مثال:-

```
#include<iostream>
using namespace std;
class example
{
public:
    int p[5];
}obj1;
void main()
{
    cout<<"Enter 5 Numbers.\n";
    for(int a=0;a<5;a++)
        cin>>obj1.p[a];
    cout<<"The Numbers after adding 2 are\n";
    for(int b=0;b<5;b++)
        cout<<obj1.p[b]+2<<" ";
}
```

سيكون الـ output كالتالي

```
Enter 5 Numbers.
21
32
23
-90
98
The Numbers after adding 2 are
23 34 25 -88 100
```

طريقة أخرى لكتابة الكود السابق

```
#include<iostream>
using namespace std;
class example
{
public:
    int p[5];
    void print();
}obj1;
void main()
{
    cout<<"Enter 5 Numbers.\n";
    for(int a=0;a<5;a++)
        cin>>obj1.p[a];
    cout<<"The Numbers after adding 2 are\n";
    obj1.print();
}
void example::print()
{
        for(int b=0;b<5;b++)
        cout<<p[b]+2<<" ";
}
```

- **classes in Arrays**

➢ يمكننا عمل objects للـ class على شكل array.

مثال:-

```
#include<iostream>
using namespace std;
class example
{
    int num;
public:
    void print();
    void set(int num1){num=num1;}
}obj[7];
void main()
{
int b=0;
    for(int a=0;a<7;a++)
    {   b+=1;
        obj[a].set(b);
    }
    obj[0].print();
    obj[6].print();
}
void example::print()
{
        cout<<"the value of the object is\n";
        cout<<num<<endl;
}
```

سيكون الـ output كالتالي

```
the value of the object is
1
the value of the object is
7
```

Consider the following declarations:

```
class xClass
{
public:
        void func();
        void print() const;
        xClass();
        xClass(int, double);
private:
        int u;
        double w;
};
```

6) How many private members does class xClass have? 2

7) Write a C++ statement that declares an object t of type xClass, and initializes the member variables of t to 20 and 35.5 respectively.

```
xClass t(20, 35.5);
```

8) Write a C++ statement that prints the values of the member variables of the object t.

```
t.print();
```

9) Write the definition of the member function print that prints the contents of u and w.

```
void xClass::print() const
{
      cout << u << w;
}
```

10) Write the definition of the constructor with parameters that assigns values to u and w.

```
xClass::xClass(int a, double b)
{
      u = a;
      w = b;
}
```

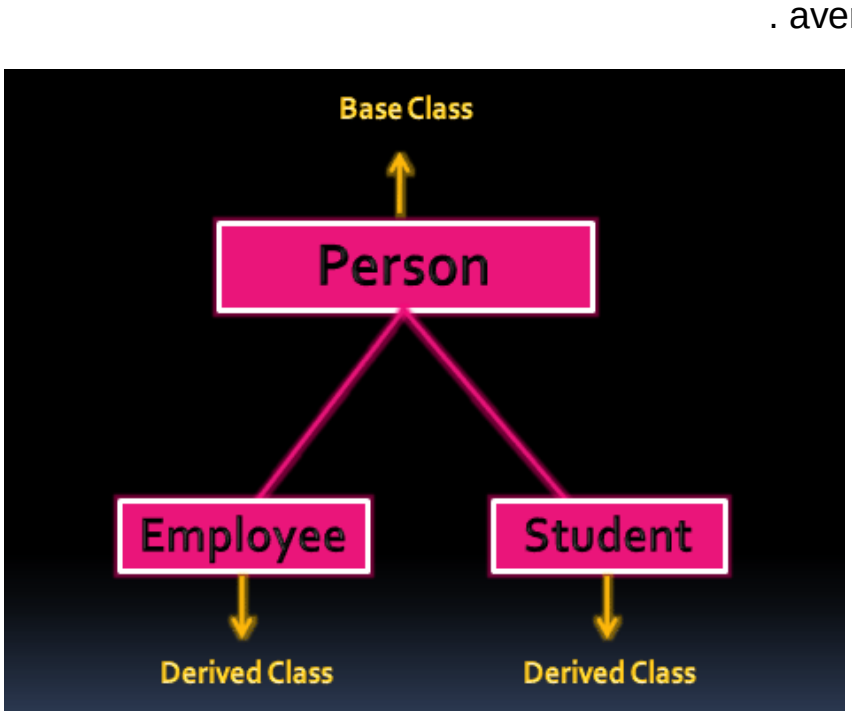
Inheritance

➢ ما المقصود بالـ inheritance ؟

هي علاقة بين 2 class أو أكثر حيثُ أنّ الـ member variables لأحد الـ classes (base class) يصبح من ضمن الـ member variables للـ class الآخر (derived class)

لتوضيح الـ inheritance

لكل إنسان له معلومات معينة خاصة به مثل الاسم و العمر ،ولكن اذا كان هذا الانسان موظف في شركة ما فسيصبح له معلومات أخرى غير الإسم و العمر مثل الـ id و الـ salary. أي أنّ له نفس الإسم و العمر و لكن أصبح له أيضاً لكونه موظف ID و salary.ولو افترضنا بأنه لدينا class اسمه A يحتوي على الإسم والعمر ولدينا class آخر يحتوي على الـ salary و الـ Id فلا يمكن أن يحتوي على إسم وعمر حتى لا يحصل إختلاف في الإسم في class A عن الإسم في class B لذلك نقوم بعمل وراثة من class A (base class) إلى class B (derived class) حيثُ في حال قمنا بتعديل الإسم و العمر عن طريق class B فإنه سيتم تغيير العمر و الإسم في class A كما أنه لو قمنا بتغيير الإسم و العمر في class A فإنه سيتغيّر في class B ،واذا كان هذا الانسان طالب سيكون له معلومات اخرى كونه طالب مثل الـ average .



o classes Inheritance

➢ يوجد 3 أنواع من الوراثة:-

1)public

2)private

3)protected

- **Public Inheritance**

➢ اذا افترضنا وجود two classes واحد بإسم (A) والآخر بإسم (B) ويوجد بينهم public Inheritance وكانت A (Base class) وكانت B (Derived class) . فإنّ

1- public member of (A) ⟶ *are Public members of (B)*

2- protected members of (A) ⟶ *are Protected members of (B)*

3- Private members of (A) ⟶ *are hidden in (B)*

أي أنّ الـ public members التابعة لـ class A أصبحت أيضاً public members لـ class B

و الـ protected members لـ class A أصبحت protected members لـ class B

و الـ private members لـ class A لا يمكن إستخدامها في class B إلا عن طريق الـ public members التابعة لـ class A .

مثال(1):-

```
#include<iostream>
#include<string>
using namespace std;
class A
{
    string name;
    int age;
public:
    void print(){cout<<"Name :"<<name<<"\nage :"<<age<<endl;}
    void read_A(){cin>>name>>age;}
    A(){name="";age=0;}//Default Constructor
    A(int a1,string str){age=a1;name=str;}//Constructor with parameters
};
class B : public A
{
    double salary;
    int ID;
public:
    void print();
    void read_B();
    B();
    B(int,string,int,double);
};
void main()
{
    B obj1;
    cout<<"### obj 1 ###\n";
    obj1.print();
    obj1.read_A();
    obj1.print();
    cout<<"### obj 2 ###\n";

    B obj2(39,"David",31,760);
    obj2.print();
    obj2.read_B();
    obj2.print();
}
void B::read_B()
{
    read_A();
    cin>>ID>>salary;
}
void B::print()
{
    cout<<"ID :"<<ID<<"\n";
        A::print();
        cout<<"Salary :"<<salary<<endl;
}
B::B()
{
    A::A();
    salary=0.0;
    ID=0;
}
B::B(int age1,string name1,int ID1,double salary1):A(age1,name1)
{
    salary=salary1;
    ID=ID1;
}
```

هكذا قمنا بعمل وراثة نوعها public
حيثُ أنّ الـ member variables الخاصة بـ A أصبحت تابعة لـ B

لقد أصبح لـ B أربع member variables وهي ID,Salary,name,age

موجود في كل من A و B و لكنه سيقوم بتنفيذ print الخاصة بـ B هو function print

B من object ولكننا إستخدمناه لـ A تابع لـ public member هو read_A

class A التابع لـ function للـ call

call للـ function التابع لـ class A ولكن لتشابه الأسماء
قمنا بوضع A:: لجملة print
وذلك من أجل تحديد من أنّ جملة الـ print هي التابعة لـ A وليست التابعة لـ B

call للـ constructor التابع لـ class A ويمكننا
كتابته بدون الـ A:: لإنه لايوجد تشابه أسماء

هكذا قمنا بإرسال هاتين القيمتين إلى الـ constructor with parametes
الخاص بـ A

لو قمنا بإرسال name ثم age فإنه لن يعمل البرنامج لأنّ الـ constructor
الخاص بـ A يأخذ أولا int ومن ثم string

```
### obj 1 ###
ID :0
Name :
age :0
Salary :0
Vendetta
39
ID :0
Name :Vendetta
age :39
Salary :0
### obj 2 ###
ID :31
Name :David
age :39
Salary :760
House
39
31
760
ID :31
Name :House
age :39
Salary :760
Press any key to continue . . .
```

obj1.print();
obj1.read_A();
obj1.print();
obj2.print();
obj2.read_B();
obj2.print();

مثال(2):-في هذا المثال سنحوّل الـ private members التابعة لـ A إلى protected members

```
#include<iostream>
#include<string>
using namespace std;
class A
{
protected:
    string name;
    int age;
public:
    void print(){cout<<"Name :"<<name<<"\nage :"<<age<<endl;}
    void read_A(){cin>>name>>age;}
    A(){name="";age=0;}//Default Constructor
    A(int a1,string str){age=a1;name=str;}//Constructor with parameters
};
class B : public A
{
    double salary;
    int ID;
public:
    void print();
    void read_B();
    B();
    B(int,string,int,double);
};
void main()
{
    B obj1;
    cout<<"### obj 1 ###\n";
    obj1.print();

    obj1.read_A();
    obj1.print();
    cout<<"### obj 2 ###\n";
    B obj2(39,"David",31,760);
    obj2.print();
    obj2.read_B();
    obj2.print();
}
void B::read_B()
{
    cin>>ID>>name>>age>>salary;
}
void B::print()
{
    cout<<"ID :"<<ID<<"\n"
        <<"Name :"<<name<<"\nage :"<<age<<endl;
        cout<<"Salary :"<<salary<<endl;
}
B::B()
{   name="";
    age=0;
    salary=0.0;
    ID=0;
}
B::B(int age1,string name1,int ID1,double salary1)
{   age=age1;
    name=name1;
    salary=salary1;
    ID=ID1;
}
```

لاحظ أنه أصبح بالإمكان إستخدام الـ protected members بدون الحاجة إلى الـ public members

إنّ الهدف من الـ protected members (تعتبر private لا يمكن إستخدامها في داخل الـ void main) هو التمكّن من إستخدام هذه الـ members بداخل الـ derived class (B) بدون أن نحتاج إلى الـ public members .

و هذا هو الإختلاف بين الـ private members و الـ protected members حيثُ أنّ الـ private members لا نستطيع الوصول إليها في الـ derived class إلا عن طريق الـ public members أما الـ protected members فيمكن إستخدامها في الـ derived class بدون الحاجة إلى الـ public members.

- **private Inheritance**

➢ اذا افترضنا وجود two classes واحد بإسم (A) والآخر بإسم (B) ويوجد بينهم private Inheritance وكانت A (Base class) وكانت B (Derived class) . فإنّ

1- public member of (A) ⟶ *are private members of (B)*

2- protected members of (A) ⟶ *are private members of (B)*

3- Private members of (A) ⟶ *are hidden in (B)*

أي أنّ الـ public members التابعة لـ class A أصبحت private members لـ class B فلا يمكن إستخدامها إلا في داخل الـ public members التابعة لـ class B

و الـ protected members لـ class A أصبحت private members لـ class B

و الـ private members لـ class A لا يمكن إستخدامها في class B إلا عن طريق الـ public members التابعة لـ class A .

مثال:-

```
#include<iostream>
#include<string>
using namespace std;
class A
{    int age;
protected:
    string name;
public:
    void print(){cout<<"\nage :"<<age<<endl;}
    void read_A(){cin>>age;}
    A(){name="";age=0;}//Default Constructor
    A(int a1,string str){age=a1;name=str;}//Constructor with parameters
};
class B : private A
{
    double salary;
    int ID;
public:
    void print();
    void read_B();
    B();
    B(int,string,int,double);
};
void main()
{
    B obj1;
    cout<<"### obj 1 ###\n";
    obj1.print();
    obj1.read_A();//invalid,becuase it;s private in class B
    obj1.print();
    cout<<"### obj 2 ###\n";

   B obj2(39,"David",31,760);
   obj2.print();
   obj2.read_B();
   obj2.print();
}
void B::read_B()
{
    read_A();   هذا الـ call من أجل إدخال قيمة لـ age لأنها ما تزال hidden عن B فلا نصل إليها إلا عن طريق الـ public members
    cin>>ID>>name>>salary;
}
void B::print()
{
    cout<<"ID :"<<ID<<"\n"
        <<"Name :"<<name<<endl
    <<"Salary :"<<salary<<endl;
    A::print();
}
B::B()
{
    A();
    salary=0.0;
    ID=0;
}
B::B(int age1,string name1,int ID1,double salary1):A(age1,name1)
{
    salary=salary1;
    ID=ID1;
}
```

إنّ name كانت protected في A ولكنها في B هي private وذلك بسبب الـ private inheritance

إنّ name كانت protected في A ولكنها في B هي private وذلك بسبب الـ private inheritance

من أجل طباعة age لأنها ما تزال hidden عن B

- **protected Inheritance**

➢ اذا افترضنا وجود two classes واحد بإسم (A) والآخر بإسم (B) ويوجد بينهم protected Inheritance وكانت A (Base class) وكانت B (Derived class) . فإنّ

1- public member of (A) ⟶ *are protected members of (B)*

2- protected members of (A) ⟶ *are protected members of (B)*

3- Private members of (A) ⟶ *are hidden in (B)*

أي أنّ الـ public members التابعة لـ class A أصبحت protected members لـ class B فلا يمكن إستخدامها إلا في داخل الـ public members التابعة لـ class B

و الـ protected members لـ class A أصبحت protected members لـ class B

و الـ private members لـ class A لا يمكن إستخدامها في class B إلا عن طريق الـ public members التابعة لـ class A .

مثال:-

```
#include<iostream>
#include<string>
using namespace std;
class A
{    int age;
protected:
    string name;
public:
    void print(){cout<<"\nage :"<<age<<endl;}
    void read_A(){cin>>age;}
    A(){name="";age=0;}//Default Constructor
    A(int a1,string str){age=a1;name=str;}//Constructor with parameters
};
class B : protected A
{
    double salary;
    int ID;
public:
    void print();
    void read_B();
    B();
    B(int,string,int,double);
};
void main()
{
    B obj1;
    cout<<"### obj 1 ###\n";
    obj1.print();
    obj1.read_A();//invalid,becuase it's protected in class B
    obj1.print();
    cout<<"### obj 2 ###\n";

     B obj2(39,"David",31,760);
     obj2.print();
     obj2.read_B();
     obj2.print();
}
void B::read_B()
{
    read_A();  من أجل الوصول إلى age لأنه hidden في B
    cin>>ID>>name>>salary;
}                          name كانت protected في A وأصبحت أيضاً protected في B
void B::print()
{
    cout<<"ID :"<<ID<<"\n" name كانت protected في A وأصبحت أيضاً protected في B
        <<"Name :"<<name<<endl
    <<"Salary :"<<salary<<endl;
    A::print();  من أجل الوصول إلى age لأنه hidden في B
}
B::B()
{
    A();
    salary=0.0;
    ID=0;
}
B::B(int age1,string name1,int ID1,double salary1):A(age1,name1)
{
    salary=salary1;
    ID=ID1;
}
```

o Composition

- إنّ الـ composition مشابه للـ inheritance ولكن طريقة كتابته تختلف عن الـ inheritance
- تكون الوراثة على شكل object من class A في داخل B class كـ member variable (public or private).

مثال:- سيكون الـ object كـ private member.

```
#include<iostream>
#include<string>
using namespace std;
class A
{    int age;
     string name;
public:
     void print(){cout<<"Name :"<<name"\nage :"<<age<<endl;}
     void read_A(){cin>>name>>age;}
     A(){name="";age=0;}//Default Constructor
     A(int a1,string str){age=a1;name=str;}//Constructor with parameters
};
class B
{
     double salary;
     int ID;
     A obj;
public:
     void print();
     void read_B();
     B();
     B(int,string,int,double);
};
void main()
{
     B obj1;
     cout<<"### obj 1 ###\n";
     obj1.print();
     obj1.read_A();//invalid
     obj1.print();
     cout<<"### obj 2 ###\n";

      B obj2(39,"David",31,760);
      obj2.print();
      obj2.read_B();
      obj2.print();
}
void B::read_B()
{
      obj.read_A();
      cin>>ID>>salary;
}
void B::print()
{
      cout<<"ID :"<<ID<<"\n"
      <<"Salary :"<<salary<<endl;
      obj.print();
}
B::B()
{
      A();
      salary=0.0;
      ID=0;
}
B::B(int age1,string name1,int ID1,double salary1):obj(age1,name1)
{
      salary=salary1;
      ID=ID1;
}
```

لم نستخدم إسم الـ class
وإنما اسم الـ object

مثال:-في هذا المثال سيكون كـ public member

```
#include<iostream>
using namespace std;
class test
{
public:
    void print(){cout<<"the numbers of the test ="<<num_of_test;}
    void set(int n){num_of_test=n;}
private:
    int num_of_test;
};
class example
{
    int num;
public:
    void print();
    void set(int num1,int num2){num=num1;test_obj.set(num2);}
    test test_obj;
}obj;
void main()
{
        obj.set(2,3);
        obj.print();
        obj.test_obj.set(5);
        obj.print();
}
void example::print()
{
    test_obj.print();
        cout<<"the value of the object is\n";
        cout<<num<<endl;
}
```

functon call لـ set الخاصة بالـ object

قمنا بإضافة object اسمه test_obj كـ public member

سيتم تخزين 2 في الـ private الخاص بـ obj

و سيتم تخزين 3 في الـ private الخاص بـ test_obj

هكذا قمنا فقط بالتعديل على الـ private الخاص بـ test_obj فقط

functon call لـ print الخاصة بالـ object

```
the numbers of the test =3
the value of the object is
2
the numbers of the test =5
the value of the object is
2
```

مثال:- في هذا المثال سيكون كـ private member

```
#include<iostream>
using namespace std;
class test
{
public:
    void print(){cout<<"the numbers of the test ="<<num_of_test<<endl;}
    void set(int n){num_of_test=n;}
private:
    int num_of_test;
};
class example
{   int num;
    test test_obj;
public:
    void print();
    void set(int num1,int num2){num=num1;test_obj.set(num2);}
    void set_obj(int w){test_obj.set(w);}

}obj;
void main()
{       obj.set(2,3);
        obj.print();
        obj.test_obj.set(5);//invalid
        obj.set_obj(5);
        obj.print();
}
void example::print()
{   test_obj.print();
        cout<<"the value of the object is\n";
        cout<<num<<endl;
}
```

أصبحت هذه الجملة خاطئة لإنه لا يمكننا إستخدام الـ private في داخل الـ mian

سنستخدم هذه الجملة عوضاً عن الجملة السابقة

يمكننا عمل وراثة على الـ struct

للمزيد من المعلومات عن الوارثة راجع الـ appendix C صفحة 107

Consider the following statements:

```
class baseClass
{
public:
     void print() const;
     baseClass(string s = "", int a = 0);
protected:
     int x;
private:
     string str;
};

class derivedClass: public baseClass
{
public:
     void print() const;
     derivedClass(string s = "", int a = 0, int b = 0);
private:
     int y;
};
```

And given the definition of `derivedClass` constructor as follows:

```
derivedClass::derivedClass(string s, int a, int b)
            :baseClass("Hello Base", a + b)
{
     y = b;
}
```

Also, given the definition of member function `print` of `baseClass` as follows:

```
void baseClass::print() const
{
     cout << x << " " << str << endl;
}
```

1. Write the definition of the `baseClass` constructor

```
baseClass::baseClass(string s, int a)

{

     str = s;

     x = a;

}
```

2. Write the definition of the member function `print` of `derivedClass`

```
void derivedClass::print() const

{

     baseClass::print();

     cout << y;

}
```

3. What is the output of this program if the following statements were executed in `main`:

```
derivedClass dObject("DDD", 3, 7);
dObject.print();
```

```
10   Hello Base

7
```

Pointers

- إنَّ فائدة الـ pointer هي عمل متغير يشير إلى موقع ذاكرة معيَن خاص بمتغير ما (كأنَّ لموقع الذاكرة متغيرين) .

o Declaring pointer Variables

- يتم تعريف الـ pointer كالتالي

```
dataType * identifier;
```

أمثلة:-

1)

`int *p;` لقد قمنا بتعريف pointer من نوع integer اسمه p.

2)

`int *p,q;` في هذا المثال p هو الـ pointer أما q فهو عبارة عن متغير نوعه integer .

3)

`int *p,*q;` في هذا المثال p و q عبارة عن pointers نوعهما integers

4)

إنَّ الـ pointer يشير إلى DataType من نفس نوعه .

```
int *p;
int num;
double dou;
p=&num; جملة صحيحة لأنّ كلاهما من data type
p=&dou; جملة خاطئة لانّ dou من data type مختلف عن p
```

المقصود في الجملة `p=&num;` دع p يشير إلى نفس موقع الذاكرة الخاص بـ num .

لو افترضنا بأنّ address of num هو 1000 و address of p هو 2000

```
num=5;
cout<<*p<<endl; //line1 سيطبع في هذه الجملة قيمة num و هي 5
cout<<num<<endl; //line2 سيطبع في هذه الجملة 5
cout<<p<<endl;  //line3 سيطبع في هذه الجملة address of num وهو 1000
cout<<&num<<endl; //line4 سيطبع في هذه الجملة address of num وهو 1000
cout<<&p<<endl;  //line 5 سيطبع في هذه الجملة address of p وهو 2000
```

المقصود في الجملة `*p=15;` هو غيِّر قيمة num إلى 15.

لو افترضنا بأنّ address of num هو 1000 و address of p هو 2000

```
*p=15;
cout<<*p<<endl; //line1 سيطبع في هذه الجملة قيمة num و هي 15
cout<<num<<endl; //line2 سيطبع في هذه الجملة 15
cout<<p<<endl;  //line3 سيطبع في هذه الجملة address of num وهو 1000
cout<<&num<<endl; //line4 سيطبع في هذه الجملة address of num وهو 1000
cout<<&p<<endl;  //line 5 سيطبع في هذه الجملة address of p وهو 2000
```

- يشيرالـ pointer إلى موقع ذاكرة واحد فقط .

```
int *p;
int q=2,w=3;
p=&w;
p=&q;
```

في هذه الجملة سيشير p إلى q فقط.

- الجملة `int *p,q;` مشابهة لـ
```
int q;
int *p=&q;
```
`p=&q;`

o Initializing Pointer Variable

➢ لو افترضنا بأنّ p عبارة عن pointer بغض النظر عن الـ dataType و أردنا من الـ pointer (لسببٍ ما) بأن يشير إلى لا شيء . فنكتب `p=NULL;` or `p=0;`

مثال:-

```
int *p,x=2;          800 هو address of p و 500 هو address of x
p=NULL;
cout<<p<<endl; 00000000 سيطبع
cout<<&p<<endl;800 سيطبع
cout<<*p<<endl; لن يعمل البرنامج بسببها
p=&x;
cout<<p<<endl;  500 سيطبع
cout<<&p<<endl;800 سيطبع
cout<<*p<<endl;   2 سيطبع
```

note
- &p → address of p
- *p → value of x
- p → address of x
- x → value of x
- &x → address of x

➢ (*&) و (&*) يلغوا بعضهم البعض .

```
cout<<*&p<<endl;
cout<<&*p<<endl;
```
سيطبع في كلا الجملتين 500

مثال:-

```
#include<iostream>
using namespace std;

void main()
{
    int x = 5;
    int *p = &x;
    cout << p << endl;
    cout << *p << endl;
    cout << &p << endl;
    cout << *&p << endl;
    cout << &*p << endl;
    cout << x << endl;
    cout << &x << endl;
}
```

سيكون الـ output كالتالي(في حال كان address of x هو 670 وكان address of p هو 800)

```
670
5
800
670
670
5
670
```

o Operations On Pointer Variables

- Assignment (=)

إنّ عملية مساواة pointer بآخر مسموحة على أنْ يكونا من نفس الـ dataType

```
string *ptr1,*ptr2;
string str="name";
ptr1=&str;
ptr2=ptr1;
cout<<*ptr2<<endl;
```
سيطبع name

لقد أصبح كل من ptr1 و ptr2 يشيران إلى نفس موقع في الذاكرة.

- Relational (==,! =)

العمليتان (==،=!) هما الوحيدتان المسموحتان من العمليات العلائقية على الـ pointer .

```
double *ptr1,*ptr2;
if(ptr1==ptr2)
if(ptr1!=ptr2)
```
يتحقق الشرط في حال كان كل من ptr1 و ptr2 يشيران إلى نفس الموقع في الذاكرة

يتحقق الشرط في حال كان ptr1 يشير إلى موقع في الذاكرة مختلف عن الذي يشير إليه ptr2

- Arithmetic (++,- -,+,-,+=,- =)

أولاً علينا معرفة sizeof() .وظيفته هي تحديد عدد الـ bytes المحجوزة للـ data type.

```
int a=2;
cout<<sizeof(a)<<endl;
cout<<sizeof(2)<<endl;
cout<<sizeof(2.5)<<endl;
```
سيطبع 4
سيطبع 4
سيطبع 8

Data type	Bytes
Char,bool	1
int	4
double	8
string	32
struct	40

مثال:- (إنَّ الـ sizeof() مهمة عند تتبع الكود من أجل تحديد الـ address)

```
int x[]={5,6,7,8},*p;
p=x;
cout<<*p<<endl;
p+=2;
cout<<"The address of x is "<<p<<" and the value is "<<*p<<endl;
```
لو افترضنا أنّ address of x هو 130

سيطبع 5

سيطبع 138 .لانه حسب الـ p+2 سيزداد الـ address بمقدار 8 لأنّ الـ integer يزداد بمقدار 4 وذلك حسب الـ sizeof

سيطبع 7

مثال:-

```
int a[3]={1,11,111};
int *p;
p=a;
cout<<*p<<endl;
p++;
cout<<*p<<endl;
p++;
cout<<*p<<endl;
```
لو افترضنا أنّ address of a هو 350

لاحظ بأننا لم نضع & لـ a لأننا نتعامل مع array

سيطبع 1 في حال طبعنا p فإنه سيكون address of a[0] وهو 350

سيطبع 11 في حال طبعنا p فإنه سيكون address of a[1] وهو 354

سيطبع 111 في حال طبعنا p فإنه سيكون address of a[2] وهو 358

جدول الأولويات:-

Operators	Associativity	Type
() []	left to right	highest
++ --	left to right	unary (postfix)
++ --	right to left	unary (prefix)
* / %	left to right	multiplicative
+ -	left to right	additive

مثال:-

```
#include<iostream>
using namespace std;

void main(){
    double a[] = {1000, 2000, 3000, 4000};
    double *ap = a;
    int *b;
    b = new int[4];
    for(int i=0; i<4; i++)
        b[i] = 10 * i + 5;
    const double *dp = ap;
    int const *bp = b;

    cout << *(dp++) << endl;
    cout << *++dp << endl;
    cout << *(a+3) << endl;
    cout << ++ap << endl;
    cout << &(*(a+2)) << endl;
    cout << *ap << endl;
    cout << dp - ap << endl;
    cout << *(++a) << endl; //invalid
    cout << *b+1 << endl;
    cout << *bp++ << endl;
    cout << *(bp++) << endl;
    cout << ++*bp << endl; //invalid
}
```

سيكون الـ output كالتالي

```
1000
3000
4000
0012FF4C
0012FF54
2000
1
6
5
15
```

o Classes, Structs ,and Pointers Variables

- struct

➢ يمكن أن نعمل object في الـ struct كـ pointer،وهذا الـ pointer يشير على object من نفس الـ struct الذي قمنا بعمل الـ pointer منه ، *لإنه كما ذكرنا سابقاً الـ pointer يشير إلى نفس نوع الـ data Type*

```
#include<iostream>
#include<string>
using namespace std;
struct studentType
{
    string fName;
    int age;
};
void main ()
{
    studentType student,*studentptr;//  struct كلاهما من نفس الـ
    studentptr=&student;
    (*studentptr).age=15;"*" أولوية اعلى من "." وضعنا الأقواس من أجل الأولويات حيثُ للـ
    cout<<student.age<<endl; 15 سيطبع

    studentptr->age=22;  struct pointers هذه طريقة أخرى لتعيين قيم في حالة الـ
    cout<<student.age<<endl; 22 سيطبع
}
```

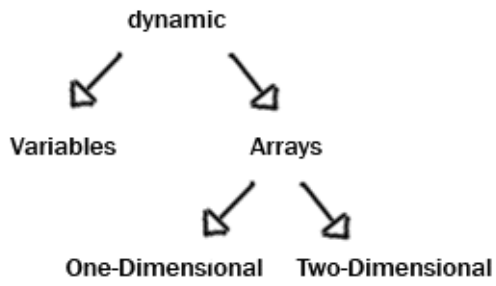
- Class

➢ يمكن أن نعمل object في الـ class كـ pointer،وهذا الـ pointer يشير على objects من نفس الـ class الذي قمنا بعمل الـ pointer منه ، *لإنه كما ذكرنا سابقاً الـ pointer يشير إلى نفس نوع الـ data Type*

```
#include<iostream>
#include<string>
using namespace std;
class ex
{
public:
    void set(int a){x=a;}
    void print(){cout<<x<<endl;}
private:
int x;
};
void main ()
{
    ex obj,* objptr;//both of them of the same class
    objptr=&obj;
    (*objptr).set(15);
    (*objptr).print();15 سيطبع
    objptr->set(22);
    obj.print();  22 سيطبع
}
```

o Dynamic Variables and Dynamic Arrays.

إن الـ Dynamic سيساعد المستخدم بتحديد حجم الـ array كما سنرى في الـ dynamic arrays .



- **Dynamic Variables: -** Variables that are created during program execution.
  - C++ provides two operators, new and delete, to create and destroy dynamic variables.

  - كلمتا new و delete محجوزتان.فلا يمكن إستخدامهما كإسم متغير .....

    - Operator new

```
new dataType ;
```

dataType يجب ان يكون الـ pointer من نفس نوع الـ

مثال:-

```
#include<iostream>
#include<string>
using namespace std;
void main ()
{
     int *p;
     string *str;
     p=new int;
     *p=19;
     str=new string;
     *str="Example";

     cout<<*str<<" "<<*p<<endl;
}
```

تعني هذه الجملة بأنّ p يشير إلى موقع في الذاكرة و لكنه من دون اسم فلا نصل إليه إلا من خلال الـ pointer

pointer من نفس نوع الـ

- Operator delete

```
delete pointerName ;
```

  - نستخدم delete لحذف موقع الذاكرة الذي يشير إليه الـ pointer .
  - الهدف منه هو حين يشير الـ pointer إلى موقع جديد في الذاكرة عوضاً عن الموقع القديم . فإنَّ الموقع القديم سيشكل عبئاً على الذاكرة (memory leak) لذلك يجب أن نحذف الموقع القديم قبل تعيين للـ pointer الموقع الجديد.

مثال:-

```
#include<iostream>
#include<string>
using namespace std;
void main ()
{
     int *p;
     p=new int;
     *p=19;
     delete p;
     p=new int;
     *p=16;
     cout<<*p<<endl;
}
```

بعد هذه الجملة. أصبحت القيمة القديمة (19) للـ pointer لا يمكن الوصول إليها لذلك أصبح من الضروري التخلص منها (delete p).

مثال(*):-

```
#include<iostream>
#include<string>
using namespace std;
void main()
{
     int *ptr1,*ptr2,*ptr3,x=2;
     ptr3=&x;
     ptr1=new int[5];
     for(int a=0;a<5;a++)
          ptr1[a]=3+a;
     ptr2=ptr1;
     cout<<ptr2[4]<<endl;
          delete ptr2;
     cout<<ptr1[4]<<endl;
     ptr2=ptr3;
     cout<<*ptr2<<endl;
}
```

كما قلنا مسبقاً وظيفة delete هي حذف الموقع الذي يشير إليه الـ pointer أي أنه سيتم حذف الـ array

سيطبع ravage data لأنه تمّ حذف كامل الـ array

سيكون الـ output كالتالي

```
7
-17891602
2
```

- **Dynamic Arrays: -** an array created during the execution of the program.

  - One – Dimensional arrays.
    - Operator new

```
new dataType[intExp] ;
```

    - Operator delete

```
delete [] pointerName ;
```

- كما قلنا سابقاً سنسمح للمستخدم بتحديد حجم الـ array عن طريق الـ pointer حيثُ سابقاً كان تحديد حجم الـ array يتم عن طريق المبرمج فقط.
- في هذا المثال سيتم تحديد حجم الـ array عن طريق المبرمج و لكن باستخدام الـ pointer .

```
#include<iostream>
#include<string>
using namespace std;
void main ()
{
     int *p;
     p=new int[2];
     *p=19;
     cout<<*p<<endl;
     p++;
     *p=12;
     cout<<*p<<endl;

}
```

يوجد طرق أخرى لتعيين قيم للـ array

```
(1
p[0]=19;
p[1]=19;
(2
for (int i=0;i<2;i++)
p[i]=19;
```

سيطبع 19 وهي قيمة p[0]

سيطبع 12 وهي قيمة p[1]

في المثال السابق حدد المبرمج حجم الـ array (2) وقام بتعيين قيم لها.

➢ في هذا المثال سنجعل المستخدم يحدد حجم الـ array.

```
#include<iostream>
#include<string>
using namespace std;
void main ()
{
     int *p,size;
     cout<<"please Enter the array size\n";
     cin>>size;
     p=new int[size];
     for(int a=0;a<size;a++)
          cin>>p[a];
     for(int b=0;b<size;b++)
          cout<<p[b]<<" ";
}
```

بعد أن قام المستخدم بإدخال عدد قمنا بتخزينه في dynamic array

- Two – Dimensional arrays

➢ سنتعلم الآن كيفية السماح للمستخدم بتحديد عدد الـ rows و الـ columns في الـ array .

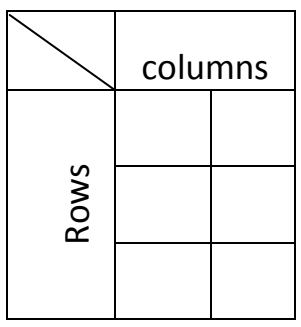
```
int **p,rows,cols;
cout<<"please Enter the array rows\n";
cin>>rows;
p=new int *[rows];
cout<<"please Enter the array columns\n";
cin>>cols;
for (int a=0;a<rows;a++)
    p[rows]=new int [cols];
for (int b=0;b<rows;b++)
    for (int c=0;c<cols;c++)
        cin>>p[b][c];

for (int d=0;d<rows;d++)
{for (int e=0;e<cols;e++)
        cout<<p[d][e]<<" ";
cout<<endl;}
```

هكذا حدد المستخدم عدد الـ rows في الـ array

هكذا حدد المستخدم عدد الـ cloumns في الـ array

لو افترضنا بأنّ المستخدم قام بإدخال عدد الـ rows (3) و عدد الـ columns (2) سيكون شكل الـ array كالتالي



و لو قام بإدخال قيم للـ array فإنه سيقوم أولاً بإدخال قيم للـ row الأول كاملاً ومن ثم الـ row الثاني ثم الثالث

o Shallow Copy versus Deep Copy

- سنتكلم عن مشكلة تواجهنا عندما نجعل 2 pointers يشيران إلى نفس الموقع في الذاكرة ، حيثُ عندما نقوم بإستعمال الـ delete لأحد الـ pointers فإنه سيتم حذف موقع الذاكرة لكلا الـ pointers

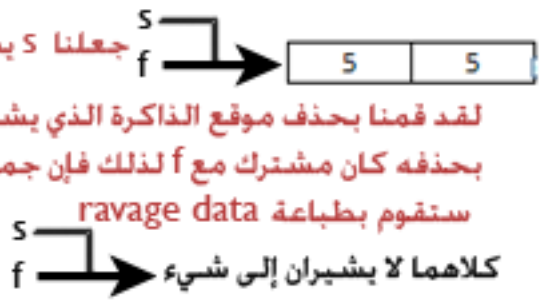
(انظر إلى المثال * صفحة 55) لقد تمّ حذف الموقع الذي يشير إليه الـ pointer ptr2 ولكنه ايضاً حذف الموقع الذي يشير إليه الـ pointer ptr1 لذلك قام بطباعة ravage data(shallow copy).

مثال:-

shallow copy

```
#include<iostream>
#include<string>
using namespace std;
void main ()
{
int *f,*s;
f=new int [2]; فمنا بعمل array تتكون من 2 rows
for(int i=0;i<2;i++)
{f[i]=5;}
s=f; جعلنا s يشير إلى نفس موقع الذاكرة الذي يشير إليه f

delete []s; لقد قمنا بحذف موقع الذاكرة الذي يشير إليه s و لكن الموقع الذي قمنا بحذفه كان مشترك مع f لذلك فإن جملة الطباعة التالية ستقوم بطباعة ravage data
for(int y=0;y<2;y++)
cout<<f[y]<<" "<<endl;
}
```



- الحل للمشكلة السابقة يكون كالتالي

Deep Copy

```
#include<iostream>
#include<string>
using namespace std;
void main ()
{
int *f,*s;
f=new int [2];
s=new int [2]; جعلنا s مشابه لـ f من حيث عدد الـ rows
for(int i=0;i<2;i++)
{f[i]=5;}
for (int k=0;k<2;k++) أصبح لـ s نفس قيم f و لكن بموقع ذاكرة مستقل عن f
{s[k]=f[k];}

delete []s; قمنا بحذف موقع الذاكرة الذي يشير إليه s
for(int y=0;y<2;y++) سيتم طباعة قيم الـ array الذي يشير إليها f
cout<<f[y]<<" "<<endl;
}
```

o Virtual functions

- في حال قمنا بعمل inheritance بين 2 classes أو 2 structs وبعد ذلك قمنا بعمل function بحيث يقوم هذا الـ function لنقل أنه سيقوم بطباعة الـ privateMembers لكل class وذلك حسب الـ object الذي نرسله إلى الـ function.(أي إذا قمنا بإرسال object من الـ baseClass فإنه سيقوم بتنفيذ print الخاصة بالـ baseClass و أما إذا أرسلنا object من الـ derivedClass فإنه سيقوم بتنفيذ print الخاصة بالـ derivedClass) سنلاحظ لأنه في الـ function protoType قمنا بتعريف object من نوع الـ baseClass كـ(parameter) `void printFunction(base & obj)` فإنه سيتم دائماً تنفيذ print الخاصة بالـ baseClass أما الخاصة بالـ derivedClass فلا يتم تنفيذها و هذه مشكلة.حتى يتم فهم هذه الفكرة انظر إلى الأمثلة.

مثال (1) :-

```
#include<iostream>
using namespace std;
class base                                  function definition
{
public:
    void print(){cout<<"In Base Class :\n"<<"a= "<<a<<"\n";}
    void set(int a1){a=a1;}
private:
    int a;
}Bobj;
class derived:public base
{
public:
    void print();
    void set(int,int,int);
private:
    int b,bb;
}Dobj;
void printFunction(base & obj){obj.print();}
void main()
{
    Bobj.set(2);
    Bobj.print();
    Dobj.set(5,6,7);
    Dobj.print();
    cout<<"After calling the function printFunction\n";
    printFunction(Bobj);
    printFunction(Dobj);
}
```

بالرغم من أننا أرسلنا object من نوع derived وكان على الـ function بطباعة نفس الـ output إلا أنه قام بطباعة قيم الـ base فقط لـ Dobj.print()

```
void derived::print()
{
    cout<<"The Derived Class\n";
    base::print();
    cout<<"In Derived class:\n"
        <<"b= "<<b<<" bb= "<<bb<<endl;
}
void derived::set(int a11,int b1,int bb1)
{
    base::set(a11);
    b=b1;
    bb=bb1;
}
```

```
In Base Class :
a= 2
The Derived Class
In Base Class :
a= 5
In Derived class:
b= 6 bb= 7
After calling the function printFunction
In Base Class :
a= 2
In Base Class :
a= 5
```

كان يجب ان يتم طباعة نفس المظلل باللون الأبيض

➢ حل المشكلة

لقد لاحظنا سابقاً بانه يتم فقط تنفيذ print الخاصة بالـ baseClass فقط

الحل للمشكلة السابقة يكون بكتابة كلمة virtual قبل الـ void print الخاصة بالـ base

```
#include<iostream>
using namespace std;
class base
{
public:
    virtual void print(){cout<<"In Base Class :\n"<<"a= "<<a<<"\n";}
    void set(int a1){a=a1;}
private:
    int a;
}Bobj;
class derived:public base
{
public:
    void print();
    void set(int,int,int);
private:
    int b,bb;
}Dobj;
void printFunction(base & obj){obj.print();}
void main()
{
    Bobj.set(2);
    Bobj.print();
    Dobj.set(5,6,7);
    Dobj.print();
    cout<<"After calling the function printFunction\n";
    printFunction(Bobj);
    printFunction(Dobj);
}
```

الآن سيتم تنفيذ print الخاصة بالـ object

الذي نقوم بإرساله إلى الـ function

```
void derived::print()
{
    cout<<"The Derived Class\n";
    base::print();
    cout<<"In Derived class:\n"
        <<"b= "<<b<<" bb= "<<bb<<endl;
}
void derived::set(int a11,int b1,int bb1)
{
    base::set(a11);
    b=b1;
    bb=bb1;
}
```

```
In Base Class :
a= 2
The Derived Class
In Base Class :
a= 5
In Derived class:
b= 6 bb= 7
After calling the function printFunction
In Base Class :
a= 2
The Derived Class
In Base Class :
a= 5
In Derived class:
b= 6 bb= 7
```

تم تنفيذ نفس المثال باللون الأبيض

مثال(2):-

```
#include<iostream>
using namespace std;
class a
{
    int aa;
public:
    void add(){aa+=2;}
    void set_a(int a1){aa=a1;}
    void print(){cout<<"a ="<<aa<<"\n";}
}Aobj;
class b:public a
{
    int bb;
public:
    void add(){a::add();bb+=2;}
    void set_ab(int b1,int a1){set_a(a1);bb=b1;}
    void print(){a::print();cout<<"b ="<<bb<<"\n";}
}Bobj;
void add_function(a& obj);
void main()
{
    Aobj.set_a(5);
    add_function(Aobj);
    Aobj.print();
    Bobj.set_ab(6,7);
    add_function(Bobj);
    Bobj.print();
    system("pause");
}
void add_function(a& obj)
{obj.add();}
```

سيكون الـ output كالتالي

```
a =7
a =9
b =6
```

لم يقم الـ function بإضافة 2 على الـ member (bb) الخاص بـالـ class b

➢ حل المشكلة

```
#include<iostream>
using namespace std;
class a
{
    int aa;
public:
   virtual void add(){aa+=2;}
    void set_a(int a1){aa=a1;}
    void print(){cout<<"a ="<<aa<<"\n";}
}Aobj;
class b:public a
{
    int bb;
public:
    void add(){a::add();bb+=2;}
    void set_ab(int b1,int a1){set_a(a1);bb=b1;}
    void print(){a::print();cout<<"b ="<<bb<<"\n";}
}Bobj;
void add_function(a& obj);
void main()
{
    Aobj.set_a(5);
    add_function(Aobj);
    Aobj.print();
    Bobj.set_ab(6,7);
    add_function(Bobj);
    Bobj.print();
    system("pause");
}
void add_function(a& obj)
{obj.add();}
```

سيكون الـ output كالتالي

```
a  =7
a  =9
b  =8
```
تم الـ function بإضافة 2 إلى bb

- Pure virtual function

➢ إذا كنا لا نريد أن نستخدم object من الـ class base يمكننا تحويل الـ function print إلى pure virtual function حينها لن نحتاج إلى كتابة الـ definition لـ function print حيث يصبح الكود

```
virtual void print (){cout<<"In base class :\n"<<a<<endl;
```
بدلاً من
```
virtual void print ()=0;
```

- Abstract Classes: a class contains one or more pure virtual functions.
  - ➢ You cannot create objects of an abstract class.

o Classes, structs and Pointers

- عند إحتواء الـ class أو الـ struct على pointer (**dynamic pointer**) كـ member variable فمن المحتمل أن نواجه مشكلة الـ memory leak أو مشكلة الـ shallow copy لذلك علينا أن نضيف إلى الـ public 3 أمور وهي Destructor و copy constructor و **(operator overload (=))سنتكلم عنها في (operator overloading).**

  - Destructor
    - Destructor automatically execute when a class goes out of scope.
    - A class can have only one destructor
    - Destructor has no parameters
  - وظيفته التخلص من مشكلة الـ memory leak .

في داخل الـ public

```
~ className () ;
```

بعد الـ void main

```
className::~className ()
{
delete [] pointerName ;
}
```

مثال:- لو قمنا بعمل objects 2 الأول هو obj1 و الثاني هو obj2 فإنّ الـ destructor ستقوم بحذف موقع الذاكرة الذي يشير إليه الـ obj2 ومن ثم الموقع الذي يشير إليه الـ obj1

```
#include<iostream>
#include<string>
using namespace std;
class student
{
public:
    student(){num=0;}
    student(int n){num=n;}
    ~student(){cout<<"the destructor delete the obj"<<num<<endl;}
private:
    int num;
};
void main()
{
    student obj1(1);
    student obj2(2);
}
```

destructor

للتأكد من أنّ الـ destructor يحذف موقع الذاكرة لآخر object قمنا بعمله قبل أول object قمنا بعمله بإستخدام الـ constructor قمنا بتخزين العدد 1 في num التابع لـ obj1 و خزّنا العدد 2 في num التابع لـ obj2

سيكون الـ output كالتالي

```
the destructor delete the obj2
the destructor delete the obj1
Press any key to continue . . .
```

- Copy Constructor

وظيفته تخزين قيم الـ member variable الخاصة بـ object بداخل object آخر من نفس الـ class

في داخل الـ public

```
className (const className& obj);
```

بعد الـ void main

```
className::className (const className & obj)
{
privateMember1=obj.privateMember1;
privateMember2=obj.privateMember2;
}
```

مثال : لتوضيح كيفية كتابة كود الـ destructor و الـ copy constructor

```
#include <iostream>
using namespace std;
class Xclass
{
public:
    void print(){for(int u=0;u<length;u++)cout<<*p<<endl;}
    void set(int *u){p=u;}
    ~Xclass();//Destructor
     Xclass (const Xclass& Object);//Copy Constructor
     Xclass (int *p1,int num){p=p1;length=num;}//constructor
private:
  int *p,length;
}obj(0,2);
void main()
{
    int a[]={1,2,3};
    int *ptr=a;
    obj.set(ptr);
    Xclass obj1(obj);
    obj1.print();
}
Xclass::~Xclass()//Destructor Definition
{
    delete [] p;
}
Xclass::Xclass(const Xclass& Object)//Copy Constructor Definition
{
    length=Object.length;
    for (int c=0;c<length;c++)
        p[c]=Object.p[c];
}
```

هكذا يتم إستخدام الـ copy constructor
حيثُ ستقوم هذه الجملة بتخزين نفس القيم للـ member variable الخاصة بـ obj في obj1

What is the output of the following programs:

```
#include <iostream>
using namespace std;
void main()
{
    const int NUM_SCORES = 3;
    int highScores[NUM_SCORES] = {15, 3, 20};

    cout << *highScores << endl;
    cout << *(highScores + 1) << endl;
    cout << *(highScores + 2) << "\n\n";
}
```

```
15

3

20
```

```
#include <iostream>
using namespace std;

void increase(int* const array, const int NUM);
void display(const int* const array, const int NUM);

void main()
{
    const int NUM_SCORES = 3;
    int highScores[NUM_SCORES] = {15, 3, 20};
    increase(highScores, NUM_SCORES);
    display(highScores, NUM_SCORES);
}
void increase(int* const array, const int NUM)
{
    for (int i = 0; i < NUM; ++i)
        array[i] += 500;
}
void display(const int* const array, const int NUM)
{
    for (int i = 0; i < NUM; ++i)
        cout << array[i] << endl;
}
```

```
515

503

520
```

```
#include <iostream>
using namespace std;

class myClass {
  int num;
public:
  void set_num(int val) {
     num = val;
  }
  void show_num(){
     cout << num << endl;
  }
};

void main()
{
  myClass ob[2], *objectPointer;
  ob[0].set_num(20);
  ob[1].set_num(30);
  objectPointer = &ob[0];
  objectPointer->show_num();
  objectPointer++;
  objectPointer->show_num();
  objectPointer--;
  objectPointer->show_num();
}
```

```
20

30

20
```

Q) What is the output of the following code:-

```
#include<iostream>
using namespace std;
void main ()
{
     int x,*p;
     int *q;
     p=new int [10];
     q=p;
     *p=4;
     for (int j=0;j<10;j++)
     {
          x=*p;
          p++;
          *p=x+j;
     }
     for (int k=0;k<10;k++)
     {
          cout<<*q<<" ";
          q++;
     }
     cout<<endl;
}
```

solution

```
4 4 5 7 10 14 19 25 32 40
```

Operator Overloading

- **this Pointer**

- لكل object(of a class or struct) له pointer يشير إلى نفسه و لكنه مخفي (hidden pointer).
- اسم الـ hidden pointer هو this.
- this كلمة محجوزة في لغة الـ C++.
- نستطيع الإستفادة من الـ hidden pointer كـ function للطباعة أو كـ function لإدخال قيم أو كـ function لـ ...
- **أهم فائدة للـ this pointer سنعرفها عندما نصل إلى overloading a unary operator.**

ثابت متغير

في داخل الـ public

```
className FunctionName ();
OR
className& FunctionName (int );
```

حيثُ أنّ هذا الـ parameter نقوم بإرساله إلى member variable سواء أكان public ام private أو protected. و يجب أن يكون من نفس الـ data type أي أنّ الذي سيستقبل هذا الـ parameter يجب أن يكون نوعه int

---

ثابت متغير

بعد الـ void main

```
ClassName ClassName::FunctionName ()
{


return *this;
}
```

أمثلة لتوضيح فكرة this pointer

(1

```
#include<iostream>
using namespace std;
class test
{
public:
    test print();
    test& cinn(int a1,int b1);
    test& set (int a2);
private:
    int a,b;
};
void main ()
{
    int s1,s2,s3;
    cout<<"Enter 2 numbers :\n";
    cin>>s1>>s2;
    example.cinn(s1,s2);
    cout<<"Enter 1 number :\n";
    cin>>s3;
    example.set(s3);
    cout<<"The numbers are :\n";
    example.print();
}
test test::print()
{
    cout<<"a ="<<a<<",b = "<<b<<endl;
    return *this;
}
test& test::cinn (int a1,int b1)
{
    a=a1;
    b=b1;
    return *this;
}
test& test::set(int a2)
{
    a=a2;
    return *this;
}
```

في هذا المثال سنستخدم الـ this pointer كـ function للطباعة و كـ function set

this pointer

انظر إلى الـ defintion لكل function

(this pointer) call

(this pointer) call

(this pointer) call

سيكون الـ output كالتالي

```
Enter 2 numbers :
2
3
Enter 1 number :
6
The Numbers are :
a=6,b=3
```

cin
cout

لقد تبيّنَ لنا من المثال السابق بأنّه يمكننا الإستغناء عن الـ functions التي كنا نستخدمها سابقاً كـ print وset وread.

(2

```
#include<iostream>
#include<string>
using namespace std;
class test
{
public:
     test print(){cout<<"a ="<<a<<" ,b ="<<b<<endl;return *this;}
     test add2(){a+=2;b=b+" world";return *this;}
     test& set (int a1,string b1){a=a1;b=b1;return *this;}
private:
     int a;
     string b;
}example;
void main ()
{
     int s1=6;
     string s2="hi";
     example.set(s1,s2);
     example.add2();
     cout<<"The output is :\n";
     example.print();
}
```

سيكون الـ output كالتالي

```
The output is :
a =8,b =hi world
```

(3

```
#include <iostream>
using namespace std;
class test
{
public:
     void print(){cout<<a<<b<<endl;}
     test set(){b='i';a='h';return *this;}
     test read(char b1,char a1){a=a1;b=b1;return *this;}
private:
     char a,b;
}example;
void main()
{
     example.set();
     example.print();
     char h,i;
     cout<<"Enter 2 letters.\n";
     cin>>h>>i;
     example.read(h,i);
     cout<<"you Entered :";
     example.print();
}
```

سيكون الـ output كالتالي

```
hi                cin
Enter 2 letters.
y
b
you Entered :by
```

- **Friend Function**

- يُسمى الـ friend function بالـ nonmember function
- ما المقصود بالـ nonmember function ؟

أي عند كتابة الـ definition له لا يحتاج إلى (className::) وعند الـ function call لا يحتاج إلى (objectName.functionName).

- للـ friend function صلاحية الوصول إلى الـ members في الـ class أو الـ struct.

```
                                  ثابت          متغير
                                في داخل الـ class
                                    يمكن أن تكون (int,bool,string,char)
friend void functionName ();
or
friend void functionName (className obj);
                                في حال أردنا إجراء تعديلات على object
```

```
                                ثابت          متغيّر
(void,int,....) functionName (className obj);
{
                                بعد الـ void main

}
```

لفهم طريقة كتابته انظر إلى الأمثلة

أمثلة لتوضيح فكرة الـ friend function

(1

```
#include<iostream>
using namespace std;
class test
{
friend void f (test obj);
public:
    void print();
private:
    int a,b;
}example;
void main ()
{   cout<<"calling The friend function \n";
    f(example);
}
void test::print()
{
    cout<<"a ="<<a<<",b = "<<b<<endl;
}
void f (test obj)
{
    obj.a=2;
    cin>>obj.b;
    cout<<"the output of the object \"Example\"\n";
    obj.print();
    test NEWobj;
    cout<<"enter a number :\n";
    cin>> NEWobj.b;
    NEWobj.a= 2 +NEWobj.b;
    cout<<"the output of the object \"NEWobj\"\n";
    NEWobj.print();
}
```

الفكرة من هذا المثال
بأنه يمكن عمل عدة امور في داخل
الـ friend function
كإدخال القيم و عمل object جديد و استدعاء function من الـ class

example (object) إلى الـ friend function قمنا بإرسال

قمنا بتحديد قيمة لـ example.a

قمنا بإدخال قيمة لـ example.b

قمنا بطباعة قيم example (object)

عملنا object اسمه NEWobj

قمنا بإدخال قيمة لـ NEWobj.b

قمنا بتحديد قيمة لـ NEWobj.a

قمنا بطباعة قيم NEWobj (object)

سيكون الـ output كالتالي

```
calling The friend function
6
the output of the object "Example"
a =2,b = 6
enter a number :
3
the output of the object "NEWobj"
a =5,b = 3
```

(2

```
#include<iostream>
using namespace std;
class test
{
friend void f ();
public:
    void print(){cout<<"a ="<<a<<",b = "<<b<<endl;}
    test(){a=1;b=2;}
private:
    int a,b;
}example;
void main ()
{   cout<<"calling The friend function \n";
    f();
}
void f ()
{
    test obj2;
    obj2=example;
    obj2.a+=6;
    obj2.print();
}
```

سيكون الـ output كالتالي

```
calling The friend function
a =7,b = 2
```

(3

```
#include<iostream>
using namespace std;
class test
{
public:
    friend void func (test obj);
    void print(){cout<<"a ="<<a<<",b ="<<b<<endl;}
    test (){a=1;b=2;}
private:
    int a,b;
};
void main ()
{   test example;
    cout<<"calling The friend function \n";
    f(example);
}
void f (test obj)
{
    test obj2;
    obj2=obj;
    obj2.a+=6;
    obj2.print();
}
```

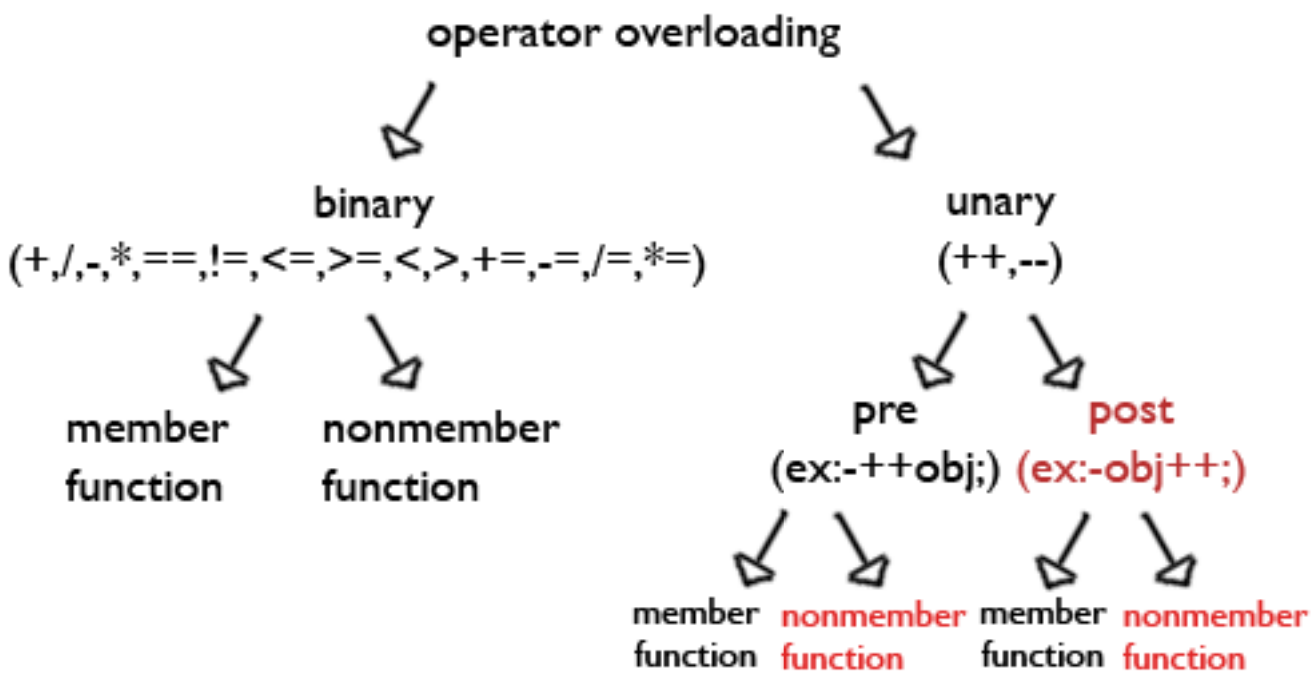
نفس الـ output المثال 2

o Operator overloading

- إنَّ العملية الوحيدة المسموحة على الـ objects من نفس الـ class أو الـ struct هي المساواة .

```
#include<iostream>
using namespace std;
class test
{
public:
     .
     .
     .
private:
     .
     .
     .
}obj1,obj2,obj3;
void main()
{
     obj1=obj2; صحيحة
                عملية المساواة مسموحة
     obj3=obj1+obj2; خاطئة
          العمليات الأخرى غير مسموحة
}
```

- العمليات الحسابية (+،-،/،*) و العمليات العلائقية (==،=!،>،<،=>،=<) غير مسموحة على الـ objects لذلك يجب أن نقوم بعملية operator overloading حتى تصبح مسموحة.



- **Binary**
  - (+,-,*,/)

➢ لكي نقوم بـ operator overloading للعمليات الحسابية علينا أن نكتب التالي.

في داخل الـ public

تُستبدل إشارة الـ # بـ + او - او * أو /

ثابت متغير

a)member function
className operator #(const className&)const;

b)nonmember function
friend className operator #(const className&,const className&);

مثال:-

```
class test
{
public:
    test operator +(const test&)const;//member
    friend test operator -(const test&,const test&);//nonmember
private:
    int a,b,c;
}obj1,obj2,obj3;
```

بعد الـ void main

تُستبدل إشارة الـ # بـ + او - او * أو /

ثابت متغير

a)member function
className className::operator #(const className& other)const
{
className temp;
temp.privateMember1=privateMember1#other.privateMember1;
temp.privateMember2=privateMember2#other.privateMember2;
return temp;
}

حسب عدد الـ private members

b)nonmember function
className operator #(const className& f,const className& s)
{
className temp;
temp.privateMember1=f.privateMember1#s.privateMember1;
temp.privateMember2=f.privateMember2#s.privateMember2;
return temp;
}

حسب عدد الـ private members

مثال:-

```
test test::operator +(const test &other) const//member
{
    test temp;
    temp.a=a+other.a;
    temp.b=b+other.b;
    temp.c=c+other.c;
    return temp;
}
test operator -(const test&f,const test&s)//non member
{
test temp;
    temp.a=f.a-s.a;
    temp.b=f.b-s.b;
    temp.c=f.c-s.c;
    return temp;
}
```

مثال:-

```
#include<iostream>
using namespace std;
class test
{public:
    void print(){cout<<"a= "<<a<<",b= "<<b<<",c= "<<c<<endl;}
    test operator +(const test&)const;//member
    friend test operator -(const test&,const test&);//nonmember
    test(){a=1;b=1;c=1;}//default constructor
    test(int a1,int b1,int c1){a=a1;b=b1;c=c1;}//constructor
private:
    int a,b,c;
}obj1,obj2(6,6,6),obj3;
void main()
{   obj3=obj1+obj2;
    obj3.print();
    obj3=obj2-obj1;
    obj3.print();
}
test test::operator +(const test &other) const//member
{   test temp;
    temp.a=a+other.a;
    temp.b=b+other.b;
    temp.c=c+other.c;
    return temp;
}
test operator -(const test&f,const test&s)//non member
 {test temp;
    temp.a=f.a-s.a;
    temp.b=f.b-s.b;
    temp.c=f.c-s.c;
    return temp;}
```

سيكون الـ output كالتالي

```
a= 7,b= 7,c= 7
a= 5,b= 5,c= 5
```

- (<=,>=,<,>)

في داخل الـ public

تُستبدل إشارة الـ # بـ =< او => او < أو >

ثابت متغير

a)member function
bool operator #(const className&)const;

b)nonmember function
friend bool operator #(const className&,const className&);

مثال:-

```
class test
{public:

    bool operator <=(const test&)const;//member
    friend bool operator >(const test&,const test&);//nonmember

private:
    int a,b,c;
}obj1,obj2(6,6,6),obj3;
```

بعد الـ void main

تُستبدل إشارة الـ # بـ =< او => او < أو >

ثابت متغير

a)member function
```
bool className::operator #(const className& other )const
{
return((privateMember1#other.privateMember1 )||
(privateMember1==other.privateMember1&&privateMember2#other.privateMember2));
}
```
b)nonmember function
```
bool operator #(const className& f,const className& s)
{
return(f.privateMember1#s.privateMember1);||
(f.privateMember==s.privateMember1&&f.privateMember2#s.privateMember2));
}
```

حسب عدد الـ private members

حسب عدد الـ private members

مثال:-

```
bool test::operator <=(const test &other) const//member
{   return ((a<=other.a)||(a==other.a&&b<=other.b)
           ||(a==other.a&&b==other.b&&c<=other.c));
}
bool operator >(const test&f,const test&s)//non member
{return ((f.a>s.a)||(f.a==s.a&&f.b>s.b)
           ||(f.a==s.a&&f.b==s.b&&f.c>s.c));
}
```

مثال

```
#include<iostream>
using namespace std;
class test
{public:
    bool operator <=(const test&)const;//member
    friend bool operator >(const test&,const test&);//nonmember
    test(){a=1;b=1;c=1;}//default constructor
    test(int a1,int b1,int c1){a=a1;b=b1;c=c1;}//constructor
    private:
    int a,b,c;
}obj1,obj2(6,6,6),obj3;
void main()
{   if(obj1<=obj3)
        cout<<"they are equal.\n";
    else cout<<"obj3 > obj1\n";
    if (obj2>obj1)
        cout<<"obj2 is bigger than obj1.\n";
    else cout<<"obj1 is bigger than obj2.\n";
}
bool test::operator <=(const test &other) const//member
{   return ((a<=other.a)||(a==other.a&&b<=other.b)
            ||(a==other.a&&b==other.b&&c<=other.c));
}
bool operator >(const test&f,const test&s)//non member
{return ((f.a>s.a)||(f.a==s.a&&f.b>s.b)
            ||(f.a==s.a&&f.b==s.b&&f.c>s.c));
}
```

سيكون الـ output كالتالي

```
they are equal.
obj2 is bigger than obj1.
```

- (=)

➢ لقد سبق أن تحدثنا في الـ pointer أنه عند إحتواء الـ class أو الـ struct على pointer(dynamic pointer) علينا أن نضيف ثلاثة أمور وهي destructor و copy constructor و الـ =

(حيثُ أنَّ الـ = تساعدنا في التخلص من مشكلة الـ shallow copy).

```
#include<iostream>
using namespace std;
class test
{public:
    const test& operator=(const test& otherList);
    void cinn(int a1, int c1){a=a1;c=c1;}
    void print() {cout<<"a= "<<a<<",c= "<<c<<endl;cout<<"b =\n";
                  for (int x=0;x<3;x++)
                  {
                      cout<<b[x]<<" ";
                  }cout<<endl;}
    int *b;
    private:
    int a,c;
}obj1,obj2;
void main()
{
    int a2,b2[]={1,2,3},c2;
    cout<<"Enter 2 Numbers :\n";
    cin>>a2>>c2;
    obj2.b=b2;
    obj2.cinn(a2,c2);
    obj1 = obj2 ;
    obj1.print();
}

const test& test::operator=(const test& otherList)
{
    if (this != &otherList)  //avoid self-assignment; Line 1
    {

        a = otherList.a;
        c = otherList.c;
        b = new int[3];
        for (int i = 0; i < 3; i++)
             b[i] = otherList.b[i];
    }
    return *this;
}
```

سيكون الـ output كالتالي

```
Enter 2 Numbers :
5
6
a= 5,c= 6
b =
1 2 3
```

- (==,!=)

في داخل الـ public

تُستبدل إشارة الـ # بـ ==

ثابت متغير

a)member function
bool operator #(const className&)const;

b)nonmember function
friend bool operator #(const className&,const className&);

مثال:-

```
class test
{public:
    bool operator==(const test&)const;//member
    friend bool operator !=(const test&,const test&);//nonmeber
    test(){a=2;c=6;}
    test (int a1,int c1){a=a1;c=c1;}
private:
    int a,c;
}obj1,obj2(7,8),obj3;
```

بعد الـ void main

تُستبدل إشارة الـ # بـ ==

ثابت متغير

a)member function
```
bool className::operator #( const className& other )const
{
return(privateMember1==other.privateMember1&&
       privateMember2==other.privateMember2);
}
```
b)nonmember function
```
bool operator #(const className& f,const className& s)
{
return(f.privateMember1==s.privateMember1&&
       f.privateMember2==s.privateMember2);
}
```

حسب عدد الـ private members

حسب عدد الـ private members

مثال:-

ملاحظة: يختلف =! عن == في الـ definition حيثُ أنَّه يتم فقط استبدال "==" بـ "=!" و بدلاً من "&&" نضع "||".

```
bool test::operator==(const test& other)const//member
{
    return (a==other.a&&c==other.c);
}
bool operator!=(const test& f,const test& s)//nonmeber
{
    return (f.a!=s.a||f.c!=s.c);
}
```

مثال:-

```
#include<iostream>
using namespace std;
class test
{public:
    bool operator==(const test&)const;//member
    friend bool operator !=(const test&,const test&);//nonmeber
    test(){a=2;c=6;}
    test (int a1,int c1){a=a1;c=c1;}
private:
    int a,c;
}obj1,obj2(7,8),obj3;
void main()
{
    if(obj1==obj3)
        cout<<"obj1 equal obj3.\n";//will execute
    else
    cout<<"obj1 not equal obj3.\n";
    if(obj1!=obj2)
        cout<<"obj1 not equal obj2.\n";//will execute
    else
        cout<<"obj1 equal obj2.\n";
}
bool test::operator==(const test& other)const//member
{
    return (a==other.a&&c==other.c);
}
bool operator!=(const test& f,const test& s)//nonmeber
{
    return (f.a!=s.a||f.c!=s.c);
}
```

سيكون الـ output كالتالي

```
obj1 equal obj3.
obj1 not equal obj2.
```

- (/=,+=,-=,*=)

في داخل الـ public

تُستبدل إشارة الـ # بـ =+ او =- او =* أو =/

ثابت متغير

a)member function

```
className operator #(const className& )const;
```

b)nonmember function

```
friend className operator #(const className&,const className&);
```

مثال:-

```
class test
{public:
    test operator+=(const test&)const;//member
    friend test operator -=(const test&,const test&);//nonmeber
    void print(){cout<<"a ="<<a<<" ,c ="<<c<<endl;}
    test(){a=2;c=6;}
    test (int a1,int c1){a=a1;c=c1;}
private:
    int a,c;
}obj1,obj2(7,7);
```

تُستبدل إشارة الـ # بـ =+ او =- او =* أو =/ بعد الـ void main

ثابت متغير

a)member function

```
className className::operator #(const className& other )const
{
className temp;
temp.privateMember1=privateMember1;temp.privateMember2=privateMember2;
temp.privateMember1=privateMember1#other.privateMember1;
temp.privateMember2=privateMember2#other.privateMember2;
return temp;
}
```

حسب عدد الـ private members

b)nonmember function

```
className operator #(const className& f,const className& s)
{
className temp;
temp.privateMember1=f.privateMember1;temp.privateMember2=f.privateMember2;
temp.privateMember1=f.privateMember1#s.privateMember1;
temp.privateMember2=f.privateMember2#s.privateMember2;
return temp;
}
```

مثال:-

```
test test::operator+=(const test& other)const//member
{    test temp;
     temp.a=a;temp.c=c;
     temp.a=a+other.a;
     temp.c=c+other.c;
     return temp;
}
test operator-=(const test& f,const test& s)//nonmeber
{    test temp;
     temp.a=f.a;temp.c=f.c;
     temp.a=f.a-s.a;
     temp.c=f.c-s.c;
     return temp;
}
```

مثال:-

```
#include<iostream>
using namespace std;
class test
{public:
    test operator+=(const test&)const;//member
    friend test operator -=(const test&,const test&);//nonmeber
    void print(){cout<<"a ="<<a<<" ,c ="<<c<<endl;}
    test(){a=2;c=6;}
    test (int a1,int c1){a=a1;c=c1;}
private:
    int a,c;
}obj1,obj2(7,7);
void main()
{
    (obj1+=obj2).print();//equal to obj1=obj1+obj2;
    (obj2-=obj1).print();//equal to obj2=obj2-obj1;
}
test test::operator+=(const test& other)const//member
{   test temp;
    temp.a=a;temp.c=c;
    temp.a=a+other.a;
    temp.c=c+other.c;
    return temp;
}
test operator-=(const test& f,const test& s)//nonmeber
{   test temp;
    temp.a=f.a;temp.c=f.c;
    temp.a=f.a-s.a;
    temp.c=f.c-s.c;
    return temp;
}
```

سيكون الـ output كالتالي

```
a =9 ,c =13
a =5 ,c =1
```

- **Unary**
  - Pre(++obj;)

في داخل الـ public

نُستبدل إشارة الـ # بـ ++ او --

ثابت متغير

a)member function
className operator #( );
b)nonmember function
friend className operator #(className&);

مثال:-

```
class test
{public:
    test operator++();//member
    friend test operator --(test&);//nonmeber
    void print(){cout<<"a ="<<a<<" ,c ="<<c<<endl;}
    test(){a=2;c=6;}
    test (int a1,int c1){a=a1;c=c1;}
private:
    int a,c;
}obj1,obj2(7,7);
```

بعد الـ void main

نُستبدل إشارة الـ # بـ + +او - -

ثابت متغير

a)member function
className className::operator #( )
{
#(privateMember1);
#(privateMember1);
return *this;
}

this pointer هنا تكون الفائدة من

private members حسب عدد الـ

b)nonmember function
className operator #(className& other)
{
#(other.privateMember1);
#(other.privateMember2);
return other;
}

private members حسب عدد الـ

مثال:-

```
test test::operator++()//member
{
    ++(a);
    ++(c);
    return *this;
}
test operator--(test& other)//nonmeber
{
    --(other.a);
    --(other.c);
    return other;
}
```

مثال:-

```
#include<iostream>
using namespace std;
class test
{public:
    test operator++();//member
    friend test operator --(test&);//nonmeber
    void print(){cout<<"a ="<<a<<" ,c ="<<c<<endl;}
    test(){a=2;c=6;}
    test (int a1,int c1){a=a1;c=c1;}
private:
    int a,c;
}obj1,obj2(7,7);
void main()
{
    ++obj1;
        obj1.print();
    --obj2;
        obj2.print();
}
test test::operator++()//member
{
    ++(a);
    ++(c);
    return *this;
}
test operator--(test& other)//nonmeber
{
    --(other.a);
    --(other.c);
    return other;
}
```

سيكون الـ output كالتالي

```
a =3 ,c =7
a =6 ,c =6
```

- Post(obj++;)

في داخل الـ public

تُستبدل إشارة الـ # بـ ++ او --

ثابت متغير

a)member function

className operator #(int);

b)nonmember function

friend className operator #(className&,int);

مثال:-

```
class test
{public:
    test operator++(int);//member
    friend test operator --(test&,int);//nonmeber
    void print(){cout<<"a ="<<a<<" ,c ="<<c<<endl;}
    test(){a=2;c=6;}
    test (int a1,int c1){a=a1;c=c1;}
private:
    int a,c;
}obj1,obj2(7,7);
```

بعد الـ void main

تُستبدل إشارة الـ # بـ + +او - -

ثابت متغير

a)member function

```
className className::operator #( int num)
{className temp=*this;
(privateMember1)#;
(privateMember1)#;
return temp;
}
```

this pointer هنا تكون الفائدة من

private members حسب عدد الـ

b)nonmember function

```
className operator #(className& other,int num2)
{className temp=other;
(temp.privateMember1)#;
(temp.privateMember2)#;
return temp;
}
```

private members حسب عدد الـ

مثال:-

```
test test::operator++(int num)//member
{    test temp=*this;
     (a)++;
     (c)++;
     return temp;
}
test operator--(test& other,int num2)//nonmeber
{    test temp=other;
     (temp.a)--;
     (temp.c)--;
     return temp;
}
```

مثال:-

```
#include<iostream>
using namespace std;
class test
{public:
    test operator++(int);//member
    friend test operator --(test&,int);//nonmeber
    void print(){cout<<"a ="<<a<<" ,c ="<<c<<endl;}
    test(){a=2;c=6;}
    test (int a1,int c1){a=a1;c=c1;}
private:
    int a,c;
}obj1,obj2(7,7);
void main()
{
    obj1++;
        obj1.print();
    obj2--;
        obj2.print();
}
test test::operator++(int num)//member
{   test temp=*this;
    (a)++;
    (c)++;
    return temp;
}
test operator--(test& other,int num2)//nonmeber
{   test temp=other;
    (temp.a)--;
    (temp.c)--;
    return temp;
}
```

سيكون الـ output كالتالي

```
a =3 ,c =7
a =6 ,c =6
```

o Templates

- إنّ الـ template مفيدة في جمع أكثر من function حيثُ أنّ هذه الـ functions تؤدي نفس الوظيفة ولكنها تختلف عن بعضها البعض بنوع الـ parameters كما أنها مفيدة في جمع أكثر من class أو struct لهم نفس الـ public و الـ private members.

- **Function template**

مثال:- في هذا المثال لدينا 3 functions تقوم بالمقارنة بين 2 parameters وطباعة الأكبر بينهما الأول يأخذ parameters من نوع integer و الثاني من نوع char و الثالث من نوع double

```
#include<iostream>
using namespace std;

int function(int,int);
char function(char,char);
double function(double,double);
void main()
{
    cout<<"The bigger between 12 and 13 is "<<function(12,13)<<endl;
    cout<<"The bigger between \'a\' and \'b\' is "<<function('a','b')<<endl;
    cout<<"The bigger between -11.5 and 13.2 is "<<function(-11.5,13.2)<<endl;

}
int function(int a,int b)
{   if(a>=b)
        return a;
    else
        return b;
}
char function(char a,char b)
{       if(a>=b)
        return a;
    else
        return b;
}
double function(double a,double b)
{   if(a>=b)
        return a;
    else
        return b;
}
```

لاحظ أنّ هذا الكود كبير ولو كان لدي برنامج يحتوي على 100 function يقومون بنفس العمل و لكنهم يختلفون بنوع الـ parameters فلن ننتهي من كتابتهم أبداً لذلك سنستخدم بما يُسمى بالـ function template حيثُ سيصبح الكود السابق كالتالي

```
#include<iostream>
using namespace std;
template<class t>
t function(t,t);
void main()
{
    cout<<"The bigger between 12 and 13 is "<<function(12,13)<<endl;
    cout<<"The bigger between \'a\' and \'b\' is "<<function('a','b')<<endl;
    cout<<"The bigger between -11.5 and 13.2 is "<<function(-11.5,13.2)<<endl;

}
template<class t>
t function( t a,t b)
{   if(a>=b)
        return a;
    else
        return b;
};
```

- **class, struct template**

➢ أمثلة

(1

```
#include<iostream>
using namespace std;
template<class type>
class test
{public:
    type larger();
    test (type a1,type c1);//constructor
private:
   type a,c;
};
void main()
{
    test<int>obj1(2,5);// integer من نوع object عملنا
    cout<<"the bigger is "<<obj1.larger()<<endl;
}
template<class type>//constructor Defintion
test<type>:: test(type a1,type c1)
{   a=a1;
    c=c1;
}
template<class type>//function Defintion
type test<type>::larger()
{   if (a>c) return a;
    else return c;
}
```

يكون function الـ template من
نوع returning value و لا يكون من
نوع void

سيكون الـ output كالتالي

```
the bigger is 5
```

(2

```
#include<iostream>
using namespace std;
template<class type>
class test
{public:
    type larger();
    test (type a1,type c1);//constructor
private:
  type  a,c;
};
void main()
{
    test<char>obj1('a','z');//   عملنا object من نوع char
    cout<<"the bigger is "<<obj1.larger()<<endl;
}
template<class type>//constructor Defintion
test<type>:: test(type a1,type c1)
{   a=a1;
    c=c1;
}
template<class type>//function Defintion
type test<type>::larger()
{   if (a>c) return a;
    else return c;
}
```

سيكون الـ output كالتالي

```
the bigger is z
```

(3

```
#include<iostream>
#include<string>
using namespace std;
template<class type>
class test
(public:
    type print();
    test (type a1,type c1);//constructor
private:
    type a,c;
);
void main()
(
    test<string>obj1("hello"," world");
    cout<<obj1.print()<<endl;
)
template<class type>//constructor Defintion
test<type>:: test(type a1,type c1)
(   a=a1;
    c=c1;
)
template<class type>//function Defintion
type test<type>::print()
(
 return a+c;
)
```

سيكون الـ output كالتالي

```
hello world
```

What is the output of the following program:

```
15 13

10 10
```

```
#include <iostream>
using namespace std;

template <class T>
class MyClass
{
  T x, y;
public:
  MyClass()
  {
     x=0;
     y=0;
  }
  MyClass(T i, T j)
  {
     x=i;
     y=j;
  }
  void getXY(T &i, T &j)
  {
     i=x;
     j=y;
  }
  MyClass operator+(MyClass object2);
  MyClass operator=(MyClass object2);
};

template <class T>
MyClass<T> MyClass<T>::operator+(MyClass<T> object2)
{
  MyClass<T> temp;
  temp.x = x + object2.x;
  temp.y = y + object2.y;
  return temp;
}

template <class T>
MyClass<T> MyClass<T>::operator=(MyClass<T> object2)
{
  x = object2.x;
  y = object2.y;
  return *this;
}

void main()
{
  MyClass<int> object1(10, 10), object2(5, 3), object3;
  int x, y;

  object3 = object1 + object2;
  object3.getXY(x, y);
  cout << x << " " << y << endl;

  object3 = object1;
  object3.getXY(x, y);
  cout << x << " " << y << endl;
}
```

Write the definition of the member function to overload the operator && for the above program

```
Template <class T>

bool MyClass<T>::operator&&(MyClass<T> object2)
{
  return (x && object2.x) && (y && object2.y);
}
```

Q) Write the definition of the function that swaps the contents of two variables.
a) Overload the operator + for the class newString to perform string concatenation. Suppose that s1 is "hello" and s2 is "world", Then the statement: s3=s1+s2; should assign "hello world" to s3, where s1, s2, and s3 are newString objects.
b) Overload the operator += for the class newString to perform string concatenation. Suppose that s1 is "hello" and s2 is "world", Then the statement: s1+=s2; should assign "hello world" to s1, where s1 and s2 are newString objects.
Solution:

```
#include<iostream>
#include<string>
using namespace std;
class newString
{   string word;
public:
    void print(){cout<<word<<"\""<<endl;}
    newString operator +(const newString&)const;
    newString operator +=(const newString&)const;
    newString(string a){word=a;}
    newString(){word="Null";}
};
void main()
{   newString s1("Hello "),s2("World"),s3;
    cout<<"the output of s3=s1+s2 is \"";
    s3=s1+s2;
    s3.print();
    cout<<"the output of s1 +=s2 is \"";
    (s1+=s2).print();
}
newString newString::operator +(const newString& other)const
{   newString temp;
    temp.word=word+other.word;
    return temp;
}
newString newString::operator +=(const newString& other)const
{   newString temp;
    temp.word=word;
    temp.word=word+other.word;
    return (temp);
}
```

سيكون الـ output كالتالي

```
the output of s3=s1+s2 is "Hello World"
the output of s1 +=s2 is "Hello World"
```

Recursive Function

- **Using Recursive function**

➢ ما المقصود بالـ Recursive Function ؟

هو function يعمل call لنفسه لعدة مرات حتى يتحقق شرط معيّن حيثُ يوجد هذا الشرط في داخل الـ function وعندما يتحقق هذا الشرط سيتوقف عن عمل الـ call لنفسه.

➢ لماذا نستخدم الـ Recursive Function ؟

نستخدم الـ Recursive Function ليكون شكل الكود واضح وبسيط للمبرمج وليس أكثر ، ولكنها تأخذ وقت أطول ، وتستهلك حجم كبير من الذاكرة الـ memory .

➢ إنَّ الـ Recursive Function يحتوي في داخله على شرطين وهما

1. Base Case:- (تعمل على ايقاف الـ Recursive Function) هو عبارة عن شرط (condition)
   يوضع داخل الـ Recursive function لكي يقوم بوقفة عن عمل call لنفسة عند قيمة معينة لانه اذا لم يتم ايقاف الـ function عن عمل call لنفسه سيؤدي ذلك الى infinite recursion ومن ثم stack Over flow Error (Run time Error) لان عند حدوث infinite سيؤدي Run out of memory
2. General Case: - هي طريقة نستخدمها داخل الـ function Recursion لكي نستطيع الوصول الى الـ Base Case (الشرط) ومن غير وجوده داخل الـ function recursion سيحدث infinite recursion داخل البرنامج .

➢ كل Recursive Function يجب أن يعمل بطريقة معينة (general case) تسمح له بالوصول الى base case

أمثلة:-

1) في هذا المثال قمنا بعمل Recursive function و لكننا لم نقم بوضع base case ولا general case **مما سيؤدي ذلك إلى حدوث infinite recursion** (أي أنّ الـ function سيعمل لنفسه call إلى المالانهاية).

```
void badPrint( int k )
{
    cout<< k ;  هذا عبارة عن call لنفس الـ Function
    badPrint( k + 1 );
}
```

2) في هذا المثال يحتوي الـ Recursive Function على base case و لكنه لا يحتوي على general case مما سيؤدي أيضاً إلى حدوث infinite recursion.

```
Base Case
void badPrint2( int k )
{
    if (k < 0) return;   Assume:
                         k is a positive integer
    cout<<k;
    badPrint2( k+1 );
}
```

3) في هذا المثال يحتوي الـ Recursive Function على base case و general case.

```
Base Case
void printInt( int k )
{
    if (k == 0) return;

    cout<< k ;
    printInt( k - 1 );  ---> General Case
}
```

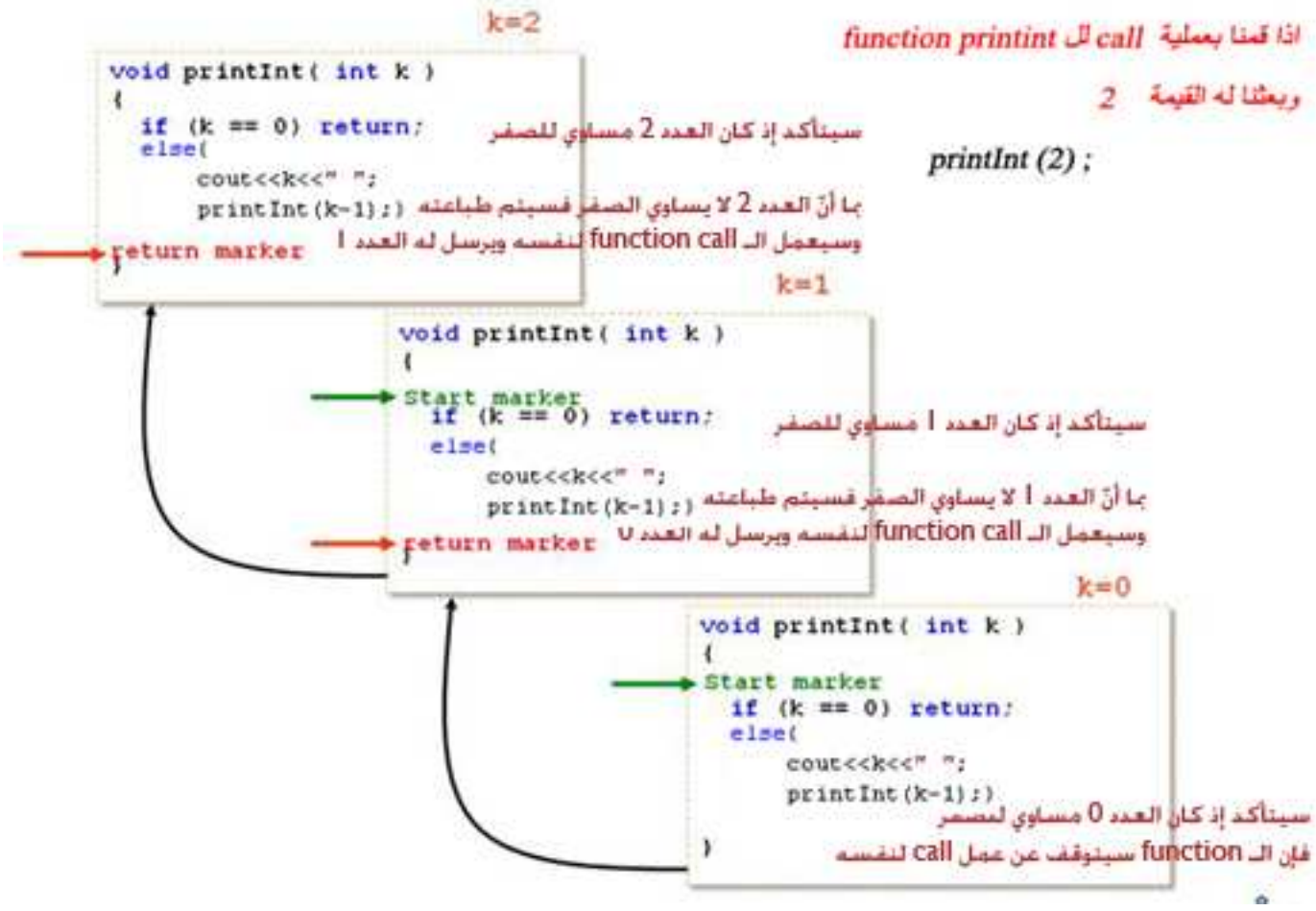
مثال(1):-

```
#include<iostream>
using namespace std;
void printInt(int k);
void main()
{
    printInt(2);
}
void printInt(int k)
{        base case
    if(k==0) return;
    else
    {   general case
        cout<<k<<" ";
        printInt(k-1);
    }
}
```

سيكون الـ output كالتالي

```
2 1
Press any key to continue . . .
```

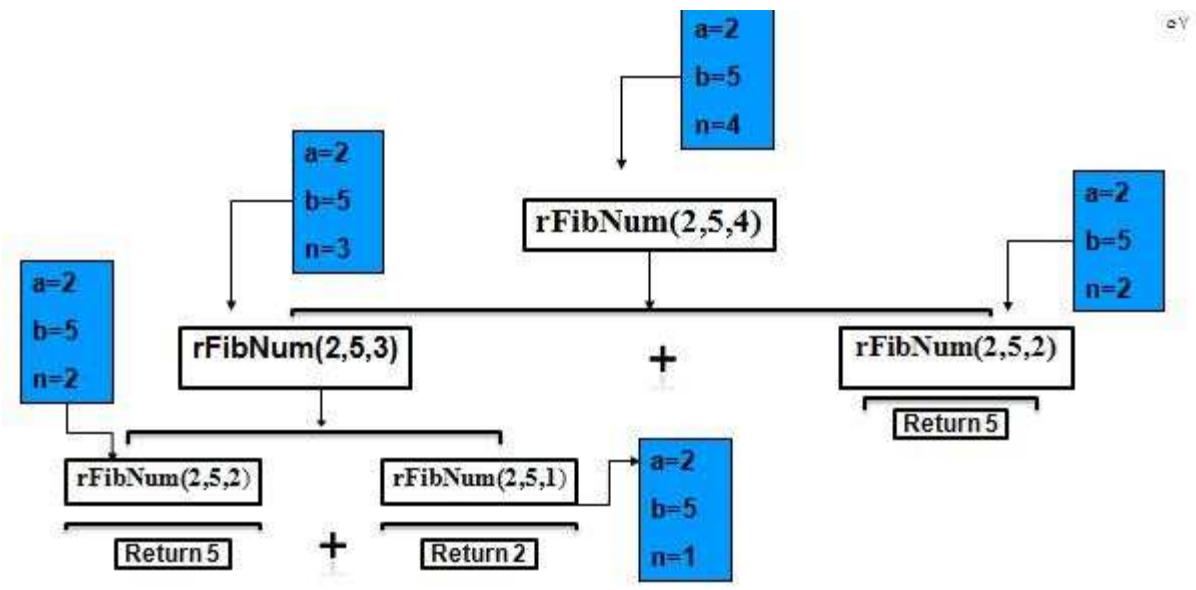
لقد قام الـ function recursion بعمله كالتالي



مثال(2):-

```
#include <iostream>
using namespace std;
int rFibNum(int a, int b, int n);
int main()
{
    int firstFibNum;
    int secondFibNum;
    int nth;
    cout << "Enter the first Fibonacci number: ";
    cin >> firstFibNum;
    cout << endl;
    cout << "Enter the second Fibonacci number: ";
    cin >> secondFibNum;
    cout << endl;
    cout << "Enter the position of the desired Fibonacci number: ";
    cin >> nth;
    cout << endl;
    cout << "The Fibonacci number at position " << nth
         << " is: " << rFibNum(firstFibNum, secondFibNum, nth)
         << endl;
    return 0;
}
int rFibNum(int a, int b, int n)
{
    if (n == 1)                    base case
        return a;
    else if (n == 2)
        return b;
    else                                    general case
        return rFibNum(a, b, n - 1) + rFibNum(a, b, n - 2);
}
```

طريقة عمله في حال أدخلنا a=2,b=5,n=4



مثال(3):-

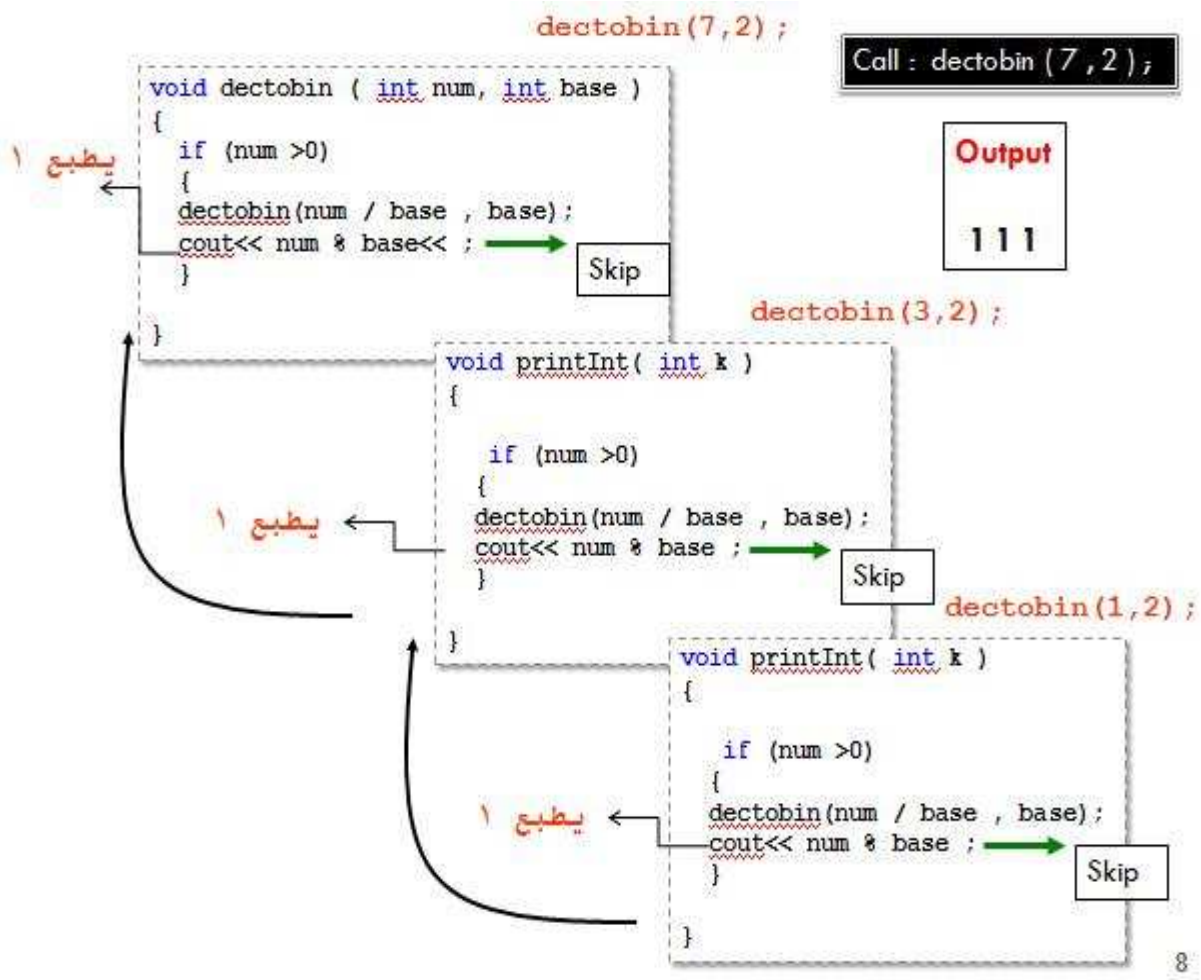
```
#include <iostream>
using namespace std;
void decToBin(int num, int base);
int main()
{
     int decimalNum;
     int base;

     base = 2;

     cout << "Enter number in decimal: ";
     cin >> decimalNum;
     cout << endl;

     cout << "Decimal " << decimalNum << " = ";
     decToBin(decimalNum, base);
     cout << " binary" << endl;
     return 0;
}
void decToBin(int num, int base)
{
     if (num > 0)
     {
          decToBin(num / base, base);
          cout << num % base;
     }
}
```

طريقة عمله في حال أدخلنا num=7 والـ base=2



مثال(4):-

```
#include <iostream>

using namespace std;

int  fact(int num);

int main()
{
    cout << "Factorial of 6 = " << fact(6) << endl;
    cout << "Factorial of 10 = " << fact(10) << endl;

    return 0;
}

int fact(int num)
{
    if (num == 0)
        return 1;
    else
        return num * fact(num - 1);
}
```

base case

general case

سيكون الـ output كالتالي

```
Factorial of 6 = 720
Factorial of 10 = 3628800
Press any key to continue . . .
```

o Function Recursion VS for loop functions

سنوضح هذا الفرق عن طريق كتابة two functions واحد باستخدام الـ for loop والاخر باستخدام الـ Recursive function لايجاد مضروب لرقم ما .

ملاحظات علي المضروب :

1- مضروب الصفر 1
2- مضروب ال N عندما (N>0) هو ..... N * N-1*N-2*N-3

Function Factorial using For loop

```
int factorial (int N)
{
    if (N == 0) return 1;
    int tmp = N;
    for (int k=N-1; k>=1; k--)
       tmp = tmp*k;

    return tmp;
}
```

اذا قمنا بعملية Call لل function factoriai
وبعثنا له القيمة 2

```
factorial (2);
```

Ouput

2

Function Factorial using Recursion

```
int fact(int num)
{
    if (num == 0)
        return 1;
    else
        return num * fact(num - 1);
}
```

Base Case

General case

اذا قمنا بعملية call للـ Function ,,,, Fact وبعثنا له القيمة 2

```
Fact(2) ;
```

The Output is : 2

نلاحظ من المثالين السابقين أنّ إستخدام الـ for Loop و إستخدام الـ Recursion يعطي نفس النتيجة لكن استخدام الـ Recursion أوضح وأبسط للمبرمج ولا يحتاج الى كتابة كثير من الكودات من استخدام الـ for Loop ولكنه في نفس الوقت أبطىء ويأخذ حجم أكثر من الذاكرة من إستخدام الـ for loop .

1) What is the output of the following program?

```
#include <iostream>
using namespace std;

void f(char *s);

void main()
{
  char str[] = "this is a test";
  f(str);
  cout<<"\n";
}

void f(char *s)
{
  if(*s)
    f(s+1);
  else
    return;

  cout << *s;
}
```

```
tset a si siht
```

2) What is the output of the following program?

```
#include <iostream>
using namespace std;

void print(int);
void main()
{
   print(12345);
}
void print(int num)
{
    if (num > 0){
        cout<<(num % 10);
        print(num / 10);
        cout<<(num % 10);
    }
}
```

```
5432112345
```

Q) Write a C++ program which let the user enter any number then print that number after reverse the number that the user entered (Using Recursive Function).

Solution:

```
#include <iostream>
#include<cmath>
using namespace std;
void reverse(int,int);
int main ()
{    int digit,num;
     cout<<"Insert the number of digits: "<<endl;
     cin>>digit;
     cout<<"Please, Insert a positive number from "<<digit<<" digits"<<endl;
     cin>>num;
     cout<<"The reverse number is ";
     reverse(digit,num);
     cout<<endl<<endl;
     return 0;
}int i=0;
void reverse (int d,int n)
{    int sol;
     if(i<d)
     {    sol=0;
          sol=n%static_cast<int>(pow(10,i+1));
          if (i==0)
          cout<<sol;
               else{
                    sol=sol/pow(10,i);
                         cout<<sol;  }
               i++;
               reverse(d,n);    }
}
```

cctype & bitwise operators

- **cctype**

➢ جميع هذه الـ functions تعمل على char

FUNCTION	DESCRIPTION	EXAMPLE

toupper(*Char_Exp*)	يقوم بتحويل من small letter إلى capital letter و يرجعه على شكل integer (ascll)	`char c = toupper('a');`
		`cout << c;`
		Outputs: A

tolower(*Char_Exp*)	يقوم بتحويل من capital letter إلى small letter و يرجعه على شكل integer (ascll)	`char c = tolower('A');`
		`cout << c;`
		Outputs: a

isupper(*Char_Exp*)	يتحقق الشرط في حال كان الحرف capital letter	`if (isupper(c))`
		`  cout << "Is uppercase.";`
		`else`
		`  cout << "Is not uppercase.";`

islower(*Char_Exp*)	يتحقق الشرط في حال كان الحرف small letter	`char c = 'a';`
		`if (islower(c))`
		`  cout << c << " is lowercase.";`
		Outputs: a is lowercase.

مثال:-

سيطبع الـ ascll لـ A و هو 65

لأننا قمنا بتخزين الـ 65 (ascll) في c فإنه سيطبع A

```
#include<iostream>
using namespace std;
int main()
{
    cout<<"\t###### toupper Section ######\n";
    cout<<"toupper('a')"<<toupper('a')<<endl;
    char c=toupper('a');
    cout<<"c="<<c<<endl;
    cout<<"\t###### tolower Section ######\n";
    cout<<"toupper('A')"<<tolower('A')<<endl;
    c=tolower('A');
    cout<<"c="<<c<<endl;
    cout<<"\t###### isupper Section ######\n";
    if(isupper(c))
        cout<<"c Is uppercase .\n";
    else
        cout<<"c Is lowercase .\n";
    cout<<"\t###### islower Section ######\n";
    c='A';
    if(islower(c))
        cout<<"c Is lowercase .\n";
    else
        cout<<"c Is uppercase .\n";
}
```

سيكون الـ output كالتالي

```
        ####### toupper Section #######
toupper('a')65
c=A
        ####### tolower Section #######
toupper('A') 97
c=a
        ####### isupper Section #######
c Is lowercase .
        ####### islower Section #######
c Is uppercase .
```

FUNCTION	DESCRIPTION	EXAMPLE

isalpha(*Char_Exp*)	يتحقق الشرط في حال كان الـ char المرسل عبارة عن حرف a-z أو A-Z	`char c = '$';`
		`if (isalpha(c))`
		`  cout << "Is a letter.";`
		`else`
		`  cout << "Is not a letter.";`
		Outputs: Is not a letter.

isdigit(*Char_Exp*)	يتحقق الشرط في حال كان الـ char المرسل عبارة عن رقم 0-9	`if (isdigit('3'))`
		`  cout << "It's a digit.";`
		`else`
		`  cout << "It's not a digit.";`
		Outputs: It's a digit.

isalnum(*Char_Exp*)	يتحقق الشرط في حال كان الـ char المرسل عبارة عن رقم أو حرف	`if (isalnum('3') && isalnum('a'))`
		`  cout << "Both alphanumeric.";`
		`else`
		`  cout << "One or more are not.";`
		Outputs: Both alphanumeric.

مثال:-

```
#include<iostream>
using namespace std;
int main()
{char c='$';
    cout<<"\t###### isalpha Section ######\n";
    if(isalpha(c))
        cout<<"c Is  a letter .\n";
    else
        cout<<"c Is not  a letter .\n";
    cout<<"\t###### isdigit Section ######\n";
    c='7';
    if(isdigit(c))
        cout<<"c Is  a digit .\n";
    else
        cout<<"c Is not  a digit .\n";
    cout<<"\t###### isalnum Section ######\n";
    char x='2';c='a';
    if(isalnum(c)&&isalnum(x))
        cout<<"both are alphanumeric.\n";
    else
        cout<<"both are not alphanumeric.\n";
}
```

سيكون الـ output كالتالي

```
        ###### isalpha Section ######
c Is not  a letter .
        ###### isdigit Section ######
c Is  a digit .
        ###### isalnum Section ######
both are alphanumeric.
```

- **bitwise operators**

➢ سنتعلم كيف نتعامل مع الأعداد بالنظام الثنائي

التحويل من النظام الثنائي إلى النظام العشري

حوّل 1101 إلى النظام العشري ؟

الحل:-

3 2 1 0
1101

$1*2^3+0*2^2+1*2^1+1*2^0$

الناتج هو

13=8+4+0+1

التحويل من النظام العشري إلى النظام الثنائي

حوّل 11 إلى النظام الثنائي ؟

الحل:-

5.5=11/2	1
2.5=5/2	1
1=2/2	0
0.5=1/2	1

الناتج هو

1011

Operator	Name	Description
&	bitwise AND	1&1=1 0&1=0 1&0=0 0&0=0
|	bitwise inclusive OR	1|1=1 0|1=1 1|0=1 0|0=0
^	bitwise exclusive OR	1^1=0 0^1=1 1^0=1 0^0=0
<< n	left shift	بعد أن يتم تحويل العدد إلى النظام الثنائي أضف العدد n من الأصفار من جهة اليمين
>> n	right shift with sign extension	بعد أن يتم تحويل العدد إلى النظام الثنائي أحذف n من الأعداد(0,1) من جهة اليمين

مثال :-

```
#include<iostream>
using namespace std;
void main()
{    cout<<"\t###### & section ######\n";
     int a=5,a1=11;
     cout<<(a & a1)<<endl;

     a=4,a1=8;
     cout<<(a & a1)<<endl;

     cout<<"\t###### | section ######\n";
     a=5,a1=11;
     cout<<(a | a1)<<endl;

     a=4,a1=8;
     cout<<(a | a1)<<endl;
     cout<<"\t###### ^ section ######\n";
     a=5,a1=11;
     cout<<(a ^ a1)<<endl;

     a=4,a1=8;
     cout<<(a ^ a1)<<endl;
     cout<<"\t###### << section ######\n";
     a=2;
     cout<<(a << 1)<<endl;
     cout<<(a << 3)<<endl;
     cout<<"\t###### >> section ######\n";
     a=12;
     cout<<(a >> 1)<<endl;
     cout<<(a >> 3)<<endl;
}
```

5 في النظام الثنائي 101

11 في النظام الثنائي 1011

صورتها في النظام الثنائي ستكون 0101 و بعدها ستحوّل إلى النظام العشري سيطبع 1

```
0101
 &
1011
```

صورتها في النظام الثنائي ستكون 0101 و بعدها ستحوّل إلى النظام 1111 العشري سيطبع 15

```
0101
 |
1011
```

صورتها في النظام الثنائي ستكون 0101 و بعدها ستحوّل إلى النظام 1110 العشري سيطبع 14

```
0101
 ^
1011
```

2 في النظام الثنائي 10

سيتم إضافة 0 إلى 10 لتصبح 100 ثم تحوّل إلى النظام العشري و سيطبع 4

سيتم إضافة 000 إلى 10 لتصبح 10000 ثم تحوّل إلى النظام العشري و سيطبع 16

12 في النظام الثنائي 1100

سيحذف خانة واحدة من 1100 من جهة اليمين ليصبح 110 و سيطبع 6

سيحذف3 خانات من 1100 من جهة اليمين ليصبح 1 و سيطبع 1

سيكون الـ output كالتالي

```
###### & section ######
1
0

###### ! section ######
15
12

###### ^ section ######
14
12

###### << section ######
4
16

###### >> section ######
6
1
```

Appendix A

Assignment, Arithmetic & Relational operations

- Assignment

لقد تعلمنا سابقاً بأنه يمكننا بأن نقوم بمساواة الـ member variables الخاصة بـ object مع object آخر بشرط أن يكونا من نفس الـ struct

لفترض بأنّ obj1 و obj2 عبارة عن objects و كانا يحتويان على str وهو عبارة عن public member variable وعلى num وهو أيضاً public member variable ونريد أن نقوم بمساواة الـ str في كل object و لكن من دون مساواة num ، لذلك فإنّ `obj2=obj1;` لن تفيدنا لأنها ستقوم بمساواة كل من str و num لذلك فسنستخدم عوضاً عنها `obj2.str=obj1.str;` حيثُ ستقوم بمساواة str فقط

مثال:-

```
#include<iostream>
#include<string>
using namespace std;
struct student
{
    string str;
    int num;
};
void main()
{
    student obj1,obj2,obj3;
    obj1.str="david";
    obj1.num=9;
    obj2.num=39;
    obj2.str=obj1.str;
    cout<<obj2.str<<" "<<obj2.num<<endl;
}
```

سيكون الـ output كالتالي

```
david 39
```

مثال:- ليس شرطاً أن يكونا من نفس الـ struct و لكن يجب أن يكونا من نفس الـ dataType

```
#include<iostream>
#include<string>
using namespace std;
struct student
{
    string str;
    int num;
}obj1;
struct studentType
{
    string str;
    string num;
}obj2;
void main()
{
    obj1.str="david";
    obj2.str=obj1.str;
    cout<<obj2.str<<" "<<obj2.num<<endl;
    obj2.num=obj1.num;//invalid
}
```

بالرغم من أنّ obj2 من struct مختلف عن obj1 إلا أنّ هذه الجملة صحيحة لأن الـ members من نفس الـ data type

هذه الجملة خاطئة لأنّ الـ data type مختلف

- Arithmetic operations

لقد تعلمنا مسبقاً بأنه لا يمكننا القيام بالعمليات الحسابية على الـ objects إلا بعد أن نقوم بعمل operator overloading و لكن يمكننا القيام بالعمليات الحسابية على public member variable

مثال:-

```
#include<iostream>
#include<string>
using namespace std;
struct student
{
    string str;
    int num;
}obj1,obj3;
struct studentType
{
    string str;
    string num;
}obj2;
void main()
{
    obj2.str="david";
    obj3.str=" Vendetta";
    obj1.str=obj2.str+obj3.str;
    cout<<obj1.str<<endl;
    obj2.num="hello";
    obj3.num=6;
    obj1.num=obj2.num+obj3.num;//invalid
}
```

هذه الجملة صحيحة لإن الـ str من نوع string في كلا الـ struct

- Relational operations

لقد تعلمنا مسبقاً بأنه لا يمكننا القيام بالعمليات العلائقية على الـ objects إلا بعد أن نقوم بعمل operator overloading و لكن يمكننا القيام بالعمليات العلائقية على public member variable

مثال:-

```
#include<iostream>
#include<string>
using namespace std;
struct student
{
    string str;
    int num;
}obj1;
struct studentType
{
    string str;
    string num;
}obj2;
void main()
{
    obj2.str="david";
    obj1.str=" Vendetta";
    if(obj2.str==obj1.str)
        cout<<"they are equal.\n";
    else
        cout<<"They are not equal.\n";
    obj2.num="hello";
    obj1.num=6;
    if(obj1.num==obj1.str)
        cout<<"they are equal.\n";
    else
        cout<<"They are not equal.\n"
}
```

جملة صحيحة لإنه بالرغم من أنّ obj2 من struct مختلف عن obj1 إلا أن الـ members من نفس الـ data type

جملة خاطئة لإن الـ members من data type مختلف

Appendix B

Assignment, Arithmetic & Relational operations

- Assignment

لقد تعلمنا سابقاً بأنه يمكننا بأن نقوم بمساواة الـ member variables الخاصة بـ object مع object آخر بشرط أن يكونا من نفس الـ class

لفترض بأنّ obj1 و obj2 عبارة عن objects و كانا يحتويان على str وهو عبارة عن public member variable وعلى num وهو أيضاً public member variable ونريد أن نقوم بمساواة الـ str في كل object و لكن من دون مساواة num ، لذلك فإنّ `obj2=obj1;` لن تفيدنا لأنها ستقوم بمساواة كل من str و num لذلك فسنستخدم عوضاً عنها `obj2.str=obj1.str;` حيثُ ستقوم بمساواة str فقط

مثال:-

```
#include<iostream>
#include<string>
using namespace std;
class  student
{ public:
    string str;
    int num;
};
void main()
{
    student obj1,obj2,obj3;
    obj1.str="david";
    obj1.num=9;
    obj2.num=39;
    obj2.str=obj1.str;
    cout<<obj2.str<<" "<<obj2.num<<endl;
}
```

سيكون الـ output كالتالي

```
david 39
```

مثال:- ليس شرطاً أن يكونا من نفس الـ class و لكن يجب أن يكونا من نفس الـ dataType

```
#include<iostream>
#include<string>
using namespace std;
 class  student
{   public:
     string str;
     int num;
}obj1;
 class  studentType
{   public:
     string str;
     string num;
}obj2;
void main()
{
    obj1.str="david";
    obj2.str=obj1.str;
    cout<<obj2.str<<" "<<obj2.num<<endl;
    obj2.num=obj1.num;//invalid
}
```

بالرغم من أنّ obj2 من class مختلف عن obj1 إلا أنّ هذه الجملة صحيحة لأن الـ members من نفس الـ data type

هذه الجملة خاطئة لأنّ الـ data type مختلف

- Arithmetic operations

لقد تعلمنا مسبقاً بأنه لا يمكننا القيام بالعمليات الحسابية على الـ objects إلا بعد أن نقوم بعمل operator overloading و لكن يمكننا القيام بالعمليات الحسابية على public member variable

مثال:-

```
#include<iostream>
#include<string>
using namespace std;
class  student
{   public:
    string str;
    int num;
}obj1,obj3;
class  studentType
{   public:
    string str;
    string num;
}obj2;
void main()
{
    obj2.str="david";
    obj3.str=" Vendetta"; هذه الجملة صحيحة لأن الـ str من نوع string في كلا الـ class
    obj1.str=obj2.str+obj3.str;
    cout<<obj1.str<<endl;
    obj2.num="hello";
    obj3.num=6;
    obj1.num=obj2.num+obj3.num;//invalid
}
```

- Relational operations

لقد تعلمنا مسبقاً بأنه لا يمكننا القيام بالعمليات العلائقية على الـ objects إلا بعد أن نقوم بعمل operator overloading و لكن يمكننا القيام بالعمليات العلائقية على public member variable

مثال:-

```
#include<iostream>
#include<string>
using namespace std;
class  student
{   public:
    string str;
    int num;
}obj1;
class  studentType
{   public:
    string str;
    string num;
}obj2;
void main()
{
    obj2.str="david";          جملة صحيحة لإنه بالرغم من أنّ obj2 من class مختلف عن obj1 إلا أن الـ members من نفس الـ data type
    obj1.str=" Vendetta";
    if(obj2.str==obj1.str)
        cout<<"they are equal.\n";
    else
        cout<<"They are not equal.\n";
    obj2.num="hello";
    obj1.num=6;          جملة خاطئة لإن الـ members مختلف من data type
    if(obj1.num==obj1.str)
        cout<<"they are equal.\n";
    else
        cout<<"They are not equal.\n"
}
```

Appendix C

Multiple Inheritances

لقد قمنا سابقاً بتعريف الـ inheritance بتعريفها على أنها علاقة بين 2 class أو أكثر و لكننا لم نتكلم إلا عن الـ inheritance بين 2 class لذلك سنقوم بالتعرّف على طريقة كتابة الكود في حال كان لدينا inheritance بين 3 classes

- Inheritance

```
#include<iostream>
#include<string>
using namespace std;
class A
{    int age;
public:
    void print(){cout<<"Age :"<<age<<endl;}
    void read(){cin>>age;}
    A(int a=0){age=a;}//constructor with default parameters
};
class B
{    string name;
public:
    void print(){cout<<"Name :"<<name<<endl;}
    void read(){cin>>name;}
    B(string str=""){name=str;}//constructor with default parameters
};
class C:public A,protected B
{    int ID;
    double salary;
public:
    void print();
    void read();
    //constructor with default parameters
    C(int id1=0,double salary1=0.0,string name1="",int age1=0);
};

void main()
{
    C obj1(123,321,"David",39);
    obj1.print();
    obj1.read();
    obj1.print();
}
void C::print()
{    cout<<"ID :"<<ID<<endl;
    B::print();
    A::print();
    cout<<"salary :"<<salary<<endl;
}
void C::read()
{    A::read();
    B::read();
    cin>>ID>>salary;
}
C::C(int id1 , double salary1 , string name1, int age1):A(age1),B(name1)
{    salary=salary1;
    ID=id1;
}
```

- composition

```
#include<iostream>
#include<string>
using namespace std;
class A
{    int age;
public:
     void print(){cout<<"Age :"<<age<<endl;}
     void read(){cin>>age;}
     A(int a=0){age=a;}//constructor with default parameters
};
class B
{    string name;
public:
     void print(){cout<<"Name :"<<name<<endl;}
     void read(){cin>>name;}
     B(string str=""){name=str;}//constructor with default parameters
};
class C
{    int ID;
     double salary;
     A objA;
     B objB;
public:
     void print();
     void read();
     //constructor with default parameters
     C(int id1=0,double salary1=0.0,string name1="",int age1=0);
};

void main()
{
     C obj1(123,321,"David",39);
     obj1.print();
     obj1.read();
     obj1.print();
}
void C::print()
{    cout<<"ID :"<<ID<<endl;
     objB.print();
     objA.print();
     cout<<"salary :"<<salary<<endl;
}
void C::read()
{    objA.read();
     objB.print();
     cin>>ID>>salary;
}
C::C(int id1 , double salary1 , string name1, int age1):objA(age1),objB(name1)
{    salary=salary1;
     ID=id1;
}
```